

ZERO-KNOWLEDGE ON THE INTERNET

A Dissertation

Presented to the Faculty of the Graduate School
of Cornell University

in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Wei-Lung Dustin Tseng

August 2011

© 2011 Wei-Lung Dustin Tseng
ALL RIGHTS RESERVED

ZERO-KNOWLEDGE ON THE INTERNET

Wei-Lung Dustin Tseng, Ph.D.

Cornell University 2011

Zero-knowledge protocols allow one party to prove the validity of a mathematical statement to another party, without revealing *any additional information*. The use of zero-knowledge in internet applications has boomed recently; this is no surprise considering that internet privacy has become such an important issue in the last few years.

The original zero-knowledge definition considers the setting where an adversarial verifier interacts only with one honest prover. In the age of the internet, however, a great number of sessions of the same protocol are executed *concurrently*. This led to the definition of *concurrent zero-knowledge* (cZK) by Dwork, Naor and Sahai (Journal of ACM, 2004). Concurrent zero-knowledge protocols are secure against adversarial verifiers who may launch a coordinated attack against multiple independent honest provers, concurrently.

Much study has already been done on the subject of cZK, resulting in a wide range of constructions under different hardness assumptions, and in different models (e.g., the plain model or with setup assumptions). Moving beyond the original focus on constructions, this thesis works on improving our understanding of cZK in three areas: *security*, *efficiency*, and *simplicity*. In part 1 we simplify and extend the current techniques to construct cZK protocols with additional security properties such as “knowledge precision”. In part 2 we present a very practical cZK protocol in the timing model. In part 3 we investigate the curious phenomenon that no known cZK protocol is public-coin.

BIOGRAPHICAL SKETCH

Dustin was born on August 25, 1985 in Ithaca, NY. He grew up in Ithaca, as well as in Cha-Yi, Taiwan, and Abbotsford, BC, Canada. In 2006, he received a B.S. from the University of British Columbia in Computer Science and Mathematics. He then returned to Ithaca and joined Cornell University for the Ph.D. program in computer science.

ACKNOWLEDGEMENTS

First and foremost, I thank Rafael Pass, my advisor. Over the years, Rafael has been an invaluable source of inspiration and motivation, of guidance and encouragement, and of wisdom and tales. I am most grateful for his patience, especially during times when I have lost focus. Without Rafael, this thesis could not be possible.

I wish to thank everyone in the Cornell crypto group; they provide a wonderful environment for research and discussion. Dearest thanks to Rachel and Muthu (Huijia Lin and Muthuramakrishnan Venkitasubramaniam); we grew together as researchers, collaborated on many projects, and picked apart each others presentations. Additionally, I thank Tom Roeder for showing me a wonderful summer at Microsoft.

I thank my committee members, Dexter Kozen, Justin Moore and Rafael Pass, for their helpful comments and suggestions during my time at Cornell. I am also grateful to Will Evans, David Kirkpatrick and Philip Loewen for fostering my interests in research while I was at UBC. I also wish to thank my other collaborators, including Kai-Min Chung, Omkant Pandey, Alon Rosen, Amit Sahai, and Douglas Wikström; I have greatly benefited from their perspectives.

I am indebted to Yisong Yue, my student mentor, who tirelessly gives me advice on courses, research, and life. Big thanks to my office-mates: Hyung-Chan An, Huijia Lin, Lonnie Princehouse, Michael Siegenthaler, and Chun-Po Wang; they make coming to the office fun.

My final thanks are reserved for my family, my dad Wen-Ching Tzeng, my mother Hsiu-Chen Tai, and my brother Kuan-Chieh Tseng, for their incredible love and unreserved support.

TABLE OF CONTENTS

| | |
|---|-----------|
| Biographical Sketch | iii |
| Acknowledgements | iv |
| Table of Contents | v |
| List of Tables | vii |
| List of Figures | viii |
| 1 Introduction | 1 |
| 1.1 Preliminaries | 6 |
| 1.1.1 Interactive Protocols | 6 |
| 1.1.2 Interactive Proofs and Arguments | 8 |
| 1.1.3 Indistinguishability | 8 |
| 1.1.4 Commitments | 9 |
| 1.2 Zero-Knowledge and Concurrency | 11 |
| 1.2.1 Concurrent Zero-Knowledge | 12 |
| 1.2.2 Parallel Zero-Knowledge | 14 |
| 1.2.3 Precision | 15 |
| 1.2.4 Related Notions | 16 |
| 1.2.5 Useful Known Constructions | 18 |
| 2 Concurrent Zero Knowledge, Revisited | 20 |
| 2.1 The Protocol | 25 |
| 2.2 The “Lazy KP” ZK simulator | 28 |
| 2.2.1 Proof Overview | 33 |
| 2.2.2 The Actual Proof | 37 |
| 2.2.3 Improving the Running Time of the Simulator | 46 |
| 2.3 Precise Concurrent Zero-Knowledge | 49 |
| 2.4 The PRS Analysis | 51 |
| 3 Concurrent Zero Knowledge in the Timing Model, Revisited | 54 |
| 3.1 The Timing Model | 60 |
| 3.1.1 Black-Box Concurrent Zero-Knowledge in the Timing Model | 62 |
| 3.2 A cZK Protocol a Simulator in the Timing Model | 62 |
| 3.2.1 The Protocol Intuition | 63 |
| 3.2.2 Description of the protocol | 68 |
| 3.2.3 Simulator Overview | 71 |
| 3.2.4 Description of the simulator | 72 |
| 3.3 Analysis of the Simulator | 74 |
| 3.3.1 Indistinguishability of the Simulation | 77 |
| 3.3.2 Running Time of the Simulator | 79 |

| | | |
|----------|---|------------|
| 4 | On the Composition of Public-Coin Zero Knowledge Protocols | 83 |
| 4.1 | Overview | 84 |
| 4.1.1 | Techniques for Theorem 21 | 86 |
| 4.1.2 | Techniques for Theorem 22 | 92 |
| 4.1.3 | Techniques for Theorem 23 | 93 |
| 4.2 | Preliminaries | 94 |
| 4.2.1 | Resettable Soundness | 95 |
| 4.2.2 | Universal Arguments | 97 |
| 4.3 | Impossibility of Public-Coin Black-Box Parallel ZK | 98 |
| 4.3.1 | Reducing to Resettable Soundness | 99 |
| 4.3.2 | Proof of Lemma 27: Resettable-Sound Proofs | 101 |
| 4.3.3 | Proof of Lemma 28: Resettable-Sound Arguments | 102 |
| 4.3.4 | On Expected Simulation | 113 |
| 4.4 | Impossibility in the Bare Public Key Model | 114 |
| 4.5 | Public-Coin Black-Box Bounded Concurrent ZK | 120 |
| 4.5.1 | The Protocol | 120 |
| 4.5.2 | Black-Box Bounded Concurrent ZK | 122 |
| 4.6 | A Non-Black-Box Public-Coin Parallel ZK Argument | 126 |
| 4.6.1 | The Protocol | 126 |
| 4.6.2 | Completeness and Soundness | 128 |
| 4.6.3 | Parallel ZK | 130 |
| 4.7 | Application to Resettable-Sound Arguments | 132 |
| | Bibliography | 135 |

LIST OF TABLES

LIST OF FIGURES

| | | |
|-----|--|-----|
| 2.1 | Concurrent ZK argument of knowledge for NP with round parameter k | 27 |
| 2.2 | A pictorial representation of the rewinding schedule of lazy-rewind. The boxes represent <i>blocks</i> , and the lines represent <i>threads</i> . If this is the top level call (i.e., $\text{lazy-rewind}(T, \emptyset, \emptyset)$), then the thicker thread is the <i>output thread</i> , whose view is the output of Sim. . . . | 31 |
| 2.3 | The recursive procedure used by Sim—the “lazy” KP simulator. | 32 |
| 2.4 | Two possible block diagrams after the swapping procedure in Claim 2 (B and B' is swapped). The main thread is shown in a thick line, and a composable block $C > B$, drawn with dashed lines, is shown in two possible configurations. | 40 |
| 2.5 | A pictorial representation of a rewinding schedule with splitting factor 3. | 47 |
| 2.6 | A pictorial representation of the original KP rewind schedule, extended from Figure 5 of [61]. We show how a rewinding interval B is related to its sibling (B'), its parent (C), and its cousins (B', B'', B'''). To swap block B with its cousin B'' , one needs to exchange the randomness used on the two highlighted thread. . . | 52 |
| 3.1 | Traditional timing delays with 1 slot | 64 |
| 3.2 | $\delta = 2\Delta/d$ gives at most d levels of nesting. | 65 |
| 3.3 | Our main techniques of restricting the nesting depth of V^* | 66 |
| 3.4 | 2 slots and t^2 penalty . Slots of nesting sessions decrease in size very quickly. | 67 |
| 3.5 | A concurrent ZK argument for NP in the timing model. | 70 |
| 3.6 | Description of our black-box ZK simulator. | 75 |
| 4.1 | In order to interact with an outside honest verifier V , the reduction T internally maintains a partial interaction between the given resetting prover, \hat{P}^* , and the (supposedly resettably-sound) verifier V_H^* . The figure captures T after Step 1 of the $i + 1^{\text{st}}$ iteration, and illustrates some of the notations we define in the analysis. | 106 |
| 4.2 | Our public-coin black-box bounded concurrent zero-knowledge protocol. | 121 |
| 4.3 | A public-coin non-black-box parallel zero-knowledge protocol. . | 127 |
| 4.4 | \mathbf{R}_S , an NP relation that extend Barak’s construction [3] for parallel repetitions. | 127 |

CHAPTER 1

INTRODUCTION

Zero-knowledge (ZK) interactive protocols, introduced by Goldwasser, Micali and Rackoff [31], are paradoxical constructs that allow one party, called the prover, to convince another party, called the verifier, of the validity of a mathematical statement $x \in L$, while providing zero additional knowledge to the Verifier. Beyond being fascinating in their own right, ZK protocols are fundamental building blocks in cryptography and have numerous applications (e.g., see [24, 68]). Formally, the zero-knowledge property requires that the view of any adversarial verifier, V , during an interaction with the prover P , can be efficiently reconstructed by some efficient *simulator*, S . A very common way of constructing zero-knowledge protocols is to construct a *universal* simulator S that can reconstruct the view of any adversary V^* , when only given *black-box* access to V^* . This type of *black-box* simulators achieves the stronger notion of *black-box* zero-knowledge.

The original ZK definition considers the scenario where an adversarial verifier interacts only with one honest prover. In the age of the internet, however, a great number of sessions of the same protocol are executed *concurrently*, rendering the definition of ZK inadequate. This led to the definition of *concurrent zero-knowledge* (cZK), first formalized by Dwork, Naor and Sahai [20]. Consider a *concurrent* adversarial verifier who may initiate multiple sessions of the same protocol with independent honest provers; the adversary has the power to arbitrarily interleave the order in which messages are exchanged among the different sessions, and can generate messages in one session based on prover responses received in others. Concurrent zero-knowledge requires that the view

of any concurrent adversarial verifier (which is considerably more complex than the view of a single interaction) can still be reconstructed by some efficient simulator.

Previous Work

When restricted to black-box simulation, Canetti, Kilian, Petrank and Rosen [14], building upon previous results [43, 65], show that **cZK** protocols must have (essentially) logarithmic round complexity. This lower bound is matched by the work of Prabhakaran, Rosen and Sahai [61], building upon previous results [64, 42]; they construct black-box **cZK** proofs for all of **NP**, based on claw-free family of permutation, with (essentially) logarithmic rounds. Outside of the “plain model”, many black-box **cZK** protocols have been constructed relying on setup assumptions (e.g., [21, 18, 13, 20]); the setup assumptions (e.g., the common reference string model, the timing model, the bare public-key model, etc.) enables these works to bypass the aforementioned lower bound and construct sub-logarithmic, even constant round, black-box concurrent zero-knowledge protocols.

Outside of black-box simulation, no **cZK** protocol is known. However, using a novel technique, Barak [3] is able to construct a constant-round non-black-box **ZK** protocol that is secure under *bounded* composition. That is, given an a priori bound on the number of concurrent sessions, there exists a **ZK** protocol that remains secure against concurrent adversarial verifiers that initiates less sessions than the given bound (in this case, the message length of Barak’s protocol depends on this bound).

Despite having many positive constructions, there are many aspects of concurrent zero-knowledge still under investigation. For example, why are all known concurrent zero-knowledge protocols private-coin, whereas the original ZK protocols [31, 28, 10] are public-coin?¹ Other questions are more open ended, such as the search for a truly practical cZK protocol; current protocols are often inefficient (e.g., protocols in the plain model requires logarithmic rounds [14]), or require costly setup assumptions (e.g., public-key registry). Additionally, there are many desirable properties of zero-knowledge protocols, such as non-malleability [19] and precision [49], that we do not fully understand in the context of cZK (Barak, Prabhakaran and Sahai [7] are able to construct concurrent non-malleable zero-knowledge arguments², but not proofs).

Overview

In this thesis we present a collection of works regarding concurrent zero-knowledge that answers some of the questions raise above. The results are grouped into three chapters, each with its own theme. Below we give an overview of each chapter; a more detailed exposition of relevant past works and techniques can be found in each chapter.

The main result of Chapter 2 is a revamped analysis of the best known black-box cZK protocol in the plain model. Based largely on the work of Prabhakaran, Rosen and Sahai [61], our analysis contains several novel technical simplifications and extensions. As a direct consequence, we are able to construct a

¹An interactive protocol is public-coin [2] if the verifier messages are simply segments of its random tape.

²An interactive protocol is an argument instead of a proof if the soundness condition only holds with respect to polynomial-time bounded provers (as opposed to all possible interactive Turing machines).

broader class of **cZK** protocols, and have more flexibility when designing the zero-knowledge simulator. Although the new analysis by itself does not establish new theorems (the new **cZK** protocols do not improve upon existing protocols constructed by [61] in terms of round complexity or assumptions), it proves to be useful in applications. For example, we show at the end of Chapter 1 how to construct *precise* **cZK** protocols [49] easily using the new analysis. The new analysis is also used in a joint work with Lin, Pass and Venkatasubramanian [44] to construct the first concurrent *non-malleable* zero-knowledge proof [19] (the first concurrent non-malleable argument is constructed in [7]), as well as the first **cZK** protocol that is simultaneously precise and concurrently non-malleable. Chapter 2 is based on the following works: [55].

Chapter 3 focuses on developing a practical **cZK** protocol. As mentioned before, in the plain model, black-box **cZK** protocols suffers from a logarithmic round-complexity lower bound (and known non-black-box techniques are inherently impractical due to the use of Karp reductions). Therefore we propose a solution in the timing model [20]. The timing model only assumes that every party has access to a stopwatch; it is one of the cheapest setup assumptions (compared to, say, the cost of maintaining a public-key registry). Previous **cZK** protocols in the timing model [20, 25] suffer from a one and only drawback that every execution of the protocols must be delayed by the maximum round trip latency of the network. (Imagine the internet, and someone with a horrid dial-up connection!) In this chapter, we show that by increasing the round complexity modestly (though still keeping the round complexity constant), the “imposed delay” on every protocol execution can be decreased “exponentially”; our **cZK** simulator and analysis in the timing model is inspired by the work of Richardson and Kilian [64] (who construct the first **cZK** protocol in the plain

model), and the work of Pass and Venkatasubramanian [59]. Additionally, we introduce a novel technique in the timing model called *eye-for-an-eye* penalties: for example, if a (possibly adversarial) verifier took one second to respond to the last prover message, then the prover would delay its next message also by one second. With eye-for-an-eye penalties, we can reduce the “imposed delay” even further, therefore eliminating the only obstacle of having a truly practical cZK protocol in the timing model. Chapter 3 is based on the following work: [56].

In Chapter 4, we investigate why all known cZK protocols are private-coin. This is so even if we consider ZK protocols that remain secure only under parallel repetition. We show that this is no accident: only trivial languages (languages in BPP) have black-box ZK protocols secure under parallel repetition. On the other hand, Barak [3] shows that if we use non-black-box simulation techniques and settle for *bounded concurrency*³, then there exists public-coin ZK protocols for all of NP. We further show that only one such “relaxation” is necessary; that is, we construct public-coin *black-box* ZK protocols secure under bounded concurrent repetition for all of NP, as well as public-coin non-black-box ZK protocols secure under *full* parallel repetition for all of NP (building upon Barak’s techniques). Together, these works complete the picture on the existence of public-coin ZK protocols secure against parallel repetitions, given the choice of black-box simulation vs. non-black-box simulation, and full concurrency vs. bounded concurrency. Chapter 4 is based on the following works: [57, 54].

³In bounded concurrency, a protocol is secure only if the number of sessions is less than an a priori bound.

1.1 Preliminaries

Given (bit) strings x and y , let $|x|$ denotes the length of string x , and $x||y$ or more simply xy denote the concatenation of x and y .

Let \mathbb{N} to denote the set of non-negative integers, \mathbb{R} to denote the set of real numbers, and $[n]$ denote the set $\{1, \dots, n\}$ of $n \in \mathbb{N}$.

A function $\nu : \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if for all $c > 0$, $\nu(n) < n^{-c}$ for sufficiently large n (i.e., $\nu \in n^{-\omega(1)}$). Given a function g , let g^k be the function computed by composing g together k times, i.e., $g^k(x) = g(g^{k-1}(x))$ and $g^0(x) = x$.

A **probabilistic polynomial time** (PPT) machine is a machine whose running time is polynomially bounded in the length of its first input. An expected PPT machine is a machine whose expected running time is polynomially bounded in the length of its first input. A non-uniform (expected or not) PPT machine additionally receives an advice string; the advice string is a function of the length of the machine's first input.

1.1.1 Interactive Protocols

An **interactive protocol** (first introduced by Goldwasser, Micali and Rackoff [31], and Babai and Moran [2]) is a pair of interactive Turing machines, $\Pi = (P, V)$, where V runs in probabilistic polynomial time (PPT). P is called the **prover**, while V is called the **verifier**. Let $\langle P, V \rangle(x)$ denote the random variable (over the randomness of P and V) representing the output of V at the end of the interaction on common input x . If additionally V receives auxiliary input z , we write

$\langle P(x), V(x, z) \rangle$ to denote the output of V . If $\langle P, V \rangle(x) = 1$, we say the verifier accepts the input x ; if $\langle P, V \rangle(x) = 0$, we say the verifier rejects the input x . Similarly, we let $\text{View}_V(P, V)(x)$ or $\text{View}_V(P(x), V(x, z))$ denote the random variable representing the **view** of V in these interactions, where the view of V includes the random tape it used and the messages it received.

We assume without loss of generality that all protocols start with a verifier message and end with a prover message. We say Π has k **rounds** if the prover and verifier each sends k messages alternately. The notation $\langle v_1, p_1, \dots \rangle$ specifies a full or partial **transcript** of Π where v denotes verifier messages and p denotes prover messages. Π is **public-coin** if the verifier messages are just disjoint segments of V 's random tape; Π is **private-coin** otherwise. In general, both the prover and verifier may abort midway in the protocol (either maliciously, or in response to invalid messages). If Π is public-coin, however, we may without loss of generality consider non-aborting verifiers (aborting verifiers may be considered to be sending the all 0's message, which is always valid).

In Chapter 4 we discuss repeating an interactive protocol in parallel. Let $\Pi^m = (P^m, V^m)$ be Π repeated in m **parallel sessions**; that is, each prover and verifier message in Π^m is just concatenation of m copies of the corresponding message in Π . We allow m to be a polynomial, which means the protocol Π^m , on input x , contains $m(|x|)$ parallel sessions. P^m and V^m follow the strategy of P and V in each session with independent randomness; in the end, V^m accepts if and only if all m sessions are accepted by V .

1.1.2 Interactive Proofs and Arguments

We state the definition of interactive proofs (introduced by Goldwasser, Micali and Rackoff [31]) and arguments (introduced by Brassard, Chaum and Crepeau [11]).

Definition 1 (Interactive Proofs and Arguments). An interactive protocol $\Pi = (P, V)$ is an *interactive proof* (resp., *interactive argument*) system for a language L if there exists a negligible function $\nu(\cdot)$ such that the following two conditions hold:

Completeness: For every $x \in L$, $\Pr[\langle P, V \rangle(x) = 1] = 1$.

Soundness: For every $x \notin L$ and every interactive Turing machine (resp., every probabilistic polynomial time) P^* , $\Pr[\langle P^*, V \rangle(x) = 1] \leq \nu(|x|)$.

1.1.3 Indistinguishability

The following definition of computational indistinguishability originates in the seminal paper of Goldwasser and Micali [30]. Let X be a countable set of strings. A **probability ensemble** indexed by X is a sequence of random variables indexed by X . Namely, any element of $A = \{A_x\}_{x \in X}$ is a random variable indexed by X .

Definition 2 (Indistinguishability). Let X and Y be countable sets. Two ensembles $\{A_{x,y}\}_{x \in X, y \in Y}$ and $\{B_{x,y}\}_{x \in X, y \in Y}$ are **computationally indistinguishable over** $x \in X$, if for every non-uniform PPT machine D (the distinguisher) whose running time is polynomial in its first input, there exists a negligible function $\nu(\cdot)$ so

that for every $x \in X, y \in Y$:

$$\left| \Pr \left[D(x, y, A_{x,y}) = 1 \right] - \Pr \left[D(x, y, B_{x,y}) = 1 \right] \right| < \nu(|x|)$$

(In the above expression, D is simply given a sample from $A_{x,y}$ and $B_{x,y}$, respectively.) $\{A_{x,y}\}_{x \in X, y \in Y}$ and $\{B_{x,y}\}_{x \in X, y \in Y}$ are said to be **statistically indistinguishable** over X if the above condition holds for all (possibly unbounded) machines D .

1.1.4 Commitments

Commitment schemes are crucial to the construction of zero-knowledge protocols. A commitment scheme enables a party, called the **sender**, to commit itself to a value to another party, the **receiver**. At first the value is hidden from the receiver; this property is called **hiding**. At a later stage when the commitment is opened, it can only reveal a single value as determined in the committing phase; this property is called **binding**. It is enough to consider bit commitments; commitment schemes that commits only to the value 0 or 1. First we define the structure of a commitment scheme.

Definition 3 (Commitment Schemes). A commitment scheme is an interactive protocol $\text{Com} = (S, R)$ with the following properties:

1. Both the sender S and the receiver R are PPT machines.
2. The commitment scheme has two stages: a commit stage and a reveal stage. In both stages, S and R receive a security parameter 1^n as common input. S additionally receives a private input $b \in \{0, 1\}$ that is the bit to be committed.

3. The commit stage results in a joint output c , called the commitment, and a private output for S , d , called the decommitment string. Without loss of generality, c can be the full transcript of the interaction between S and R , and d can be the private random tape of S .
4. In the reveal stage, sender S sends the pair (b, d) to the receiver R . The receiver R accepts or rejects the decommitment (c, b, d) .

If S and R do not deviate from the protocol, then R should accept (with probability 1) during the reveal stage. A commitment scheme is *public-coin* if all messages sent by the receiver R are segments of its random tape.

Next we define the binding and hiding property of a commitment scheme.

Definition 4 (Binding). A commitment scheme $\text{Com} = (S, R)$ is statistically (resp. computationally) binding if for every machine (resp. non-uniform PPT machine) S^* (a malicious sender), there exists a negligible function ν such that S^* succeeds in the following game with probability at most $\nu(n)$:

On security parameter 1^n , S^* first interacts with R in the commit stage to produce commitment c . Then S outputs two decommitments $(c, 0, d_0)$ and $(c, 1, d_1)$, and succeeds if R accepts both decommitments.

The commitment scheme is perfectly binding if no machine S^* can ever succeed at the above game.

Definition 5 (Hiding). A commitment scheme $\text{Com} = (S, R)$ is computationally (resp. statistically or perfectly) hiding if for every non-uniform PPT machine (resp. every machine) R^* (a malicious receiver), the following ensembles are

computationally indistinguishable over $n \in \mathbb{N}$ (resp. statistically indistinguishable over $n \in \mathbb{N}$, or identical):

$$\{\text{View}_{R^*}(S(1^n, 0), R^*(1^n))\}_{n \in \mathbb{N}} \approx \{\text{View}_{R^*}(S(1^n, 1), R^*(1^n))\}_{n \in \mathbb{N}}$$

Non-interactive statistically-binding commitment schemes can be constructed using any one-to-one one-way function (see Section 4.4.1 of [24]). Allowing some minimal interaction (in which the receiver first sends a single random initialization message), statistically-binding commitment schemes can be obtained from any one-way function [50, 35]. Polynomial-round statistically-hiding commitment schemes can be constructed from one-way functions [33] (constant-round constructions exist under stronger assumptions).

1.2 Zero-Knowledge and Concurrency

An interactive proof is said to be **zero-knowledge** if it yields nothing beyond the validity of the statement being proved. Formally, zero-knowledge requires that the view of any adversarial verifier can be reconstructed by an efficient simulator.

Definition 6 (Zero-Knowledge [31]). An interactive protocol (P, V) for language L is zero-knowledge if for every PPT adversarial verifier V^* , there exists an expected PPT simulator S such that the following two ensembles are computationally indistinguishable over $x \in L$:

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S(x, z)\}_{x \in L, z \in \{0,1\}^*}$$

A stronger definition is that of **black-box zero-knowledge**, in which there

is one universal simulator S that must generate the view of any adversarial V^* , given only black-box access to V^* .

Definition 7 (Black-Box Zero-Knowledge [29]). An interactive protocol (P, V) for language L is black-box zero-knowledge if there exists an expected PPT simulator S such that for every PPT adversarial verifier V^* , the following two ensembles are computationally indistinguishable over $x \in L$:

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S^{V^*(x,z)}(x)\}_{x \in L, z \in \{0,1\}^*}$$

Note that the simulator S does not have access to z , the auxiliary input of V^* . As a result, we may assume that V^* is deterministic, because V^* can treat part of z as its random tape.

1.2.1 Concurrent Zero-Knowledge

Given an interactive protocol (P, V) and a polynomial m , an m -session **concurrent adversarial verifier** V^* is a PPT machine that, on common input x and auxiliary input z , interacts with up to $m(|x|)$ independent copies of P concurrently. The different interactions are called **sessions**. There are no restrictions on how V^* schedules the messages among the different sessions, and V^* may choose to abort some sessions but not others (unless if the protocol is public-coin).

Definition 8 (Concurrent Zero-Knowledge [20]). An interactive protocol (P, V) for language L is black-box concurrent zero-knowledge if for every concurrent adversarial verifier V^* (i.e., any m -session concurrent adversarial verifier for any polynomial m), there exists an expected PPT black-box simulator S such that for every common input x , auxiliary input z , the following two ensembles are

computationally indistinguishable over $x \in L$

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S(x, z)\}_{x \in L, z \in \{0,1\}^*}$$

We may also extend the definition of *black-box* zero-knowledge to the concurrent setting.

Definition 9 (Black-Box Concurrent Zero-Knowledge [20]). An interactive protocol (P, V) for language L is black-box concurrent zero-knowledge if for all polynomials m , there exists an expected PPT black-box simulator S_m such that for every common input x , auxiliary input z , and m -session concurrent adversarial verifier V^* , the following two ensembles are computationally indistinguishable over $x \in L$:

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S_m^{V^*(x,z)}(x)\}_{x \in L, z \in \{0,1\}^*}$$

As before, in the case of black-box simulation, we may assume that V^* is deterministic. Note that in order to simulate the view of an m -session concurrent adversarial verifier, the simulator must have running time at least $m(|x|)$. This is why we allow a different simulator S_m for each polynomial m .

A related notion is **bounded concurrent zero-knowledge**, where the protocol is only secure if the number of concurrent sessions is less than some a priori bound.

Definition 10 (Bounded Concurrent Zero-Knowledge [3]). Let $\Pi = (P, V)$ be an interactive protocol for a language L , and let m be a polynomial. Π is m -bounded concurrent zero-knowledge if for any m -session concurrent adversarial verifier V^* , there exists an expected PPT black-box simulator S such that for every common input x , auxiliary input z , the following two ensembles are computationally

indistinguishable over $x \in L$:

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S(x, z)\}_{x \in L, z \in \{0,1\}^*}$$

Definition 11 (Black-Box Bounded Concurrent Zero-Knowledge). Let $\Pi = (P, V)$ be an interactive protocol for a language L , and let m be a polynomial. Π is black-box m -bounded concurrent zero-knowledge if there exists an expected PPT black-box simulator S such that for every common input x , auxiliary input z , and m -session concurrent adversarial verifier V^* , the following two ensembles are computationally indistinguishable over $x \in L$:

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S^{V^*(x,z)}(x)\}_{x \in L, z \in \{0,1\}^*}$$

It goes without saying the bounded concurrency is a weaker security guarantee than normal (full) concurrency.

1.2.2 Parallel Zero-Knowledge

Sometimes we are content with zero-knowledge protocols that stay secure under parallel composition (instead of full concurrent composition). An m -session **parallel adversarial verifier** V^* is PPT machine that, on common input x and auxiliary input z , interacts with $m(|x|)$ independent **sessions** of P in parallel (i.e., V^* interacts with P^m). While V^* must schedule the messages of different sessions in parallel, V^* may still choose to abort in some sessions while continuing the protocol in other sessions (unless the protocol is public-coin). Equivalently, we may view V^* as a standard (single-session) adversarial verifier for the protocol $\Pi^m = (P^m, V^m)$.

Definition 12 (Parallel Zero-knowledge). An interactive protocol $\Pi = (P, V)$ for language L is parallel zero-knowledge (resp. black-box parallel zero-knowledge) if for every polynomial m , the protocol Π^m is zero-knowledge (resp. black-box zero-knowledge).

Definition 13 (Bounded Parallel Zero-knowledge). Given a polynomial m , an interactive protocol $\Pi = (P, V)$ for language L is m -bounded parallel zero-knowledge (resp. black-box parallel zero-knowledge) if the protocol Π^m is zero-knowledge (resp. black-box zero-knowledge).

It goes without saying that parallel zero-knowledge is a weaker security guarantee than concurrent zero-knowledge.

1.2.3 Precision

Given an interactive Turing machine M and a view \mathcal{V} of M , let $\text{Time}_M(\mathcal{V})$ denote the running time of M on view \mathcal{V} .

Definition 14 (Precise Concurrent Zero-Knowledge [49, 51]). An interactive protocol (P, V) for language L is precise concurrent zero-knowledge with precision $p(n, t)$ if there exists a polynomial q (simulation overhead) such that for every PPT adversarial verifier V^* , there exists an expected PPT simulator S such that the following two properties holds:

Zero-Knowledge: The following two ensembles are computationally indistinguishable over $x \in L$:

$$\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*} \approx \{S(x, z)\}_{x \in L, z \in \{0,1\}^*}$$

Precision: Let S_r denote running the simulator S on random tape r , and let $\text{Time}_{S_r(x,z)}$ be the running time of S_r on input x and z . Then for all $x \in L$, $z \in \{0, 1\}^*$ and sufficiently long $r \in \{0, 1\}^*$,

$$\text{Time}_{S_r(x,z)} \leq p(|x|, \text{Time}_{V^*}(S_r(x, z))) + q(|x|)$$

1.2.4 Related Notions

The following types of protocols are deeply related to zero-knowledge.

Witness Indistinguishability

Witness indistinguishable protocols are not fully zero-knowledge; it is enough that the verifier cannot distinguish between different witnesses used by the prover. In this context, we focus on languages $L \in \text{NP}$ with a corresponding witness relation \mathbf{R}_L . Namely, we consider interactions in which on common input x the prover is given a witness in $\mathbf{R}_L(x)$. For any adversarial verifier V^* , let $\text{View}_{V^*}(P(x, w), V(x, z))$ be the random variable that denotes V^* 's view in an interaction with P , when V^* is given auxiliary input z , P is given witness w , and both parties are given common input x .

Definition 15 (Witness-indistinguishability). An interactive protocol (P, V) for $L \in \text{NP}$ is computationally (resp. statistically or perfectly) witness indistinguishable (WI) for witness relation \mathbf{R}_L if for every PPT adversarial verifier V^* , and for every two sequences $\{w_x^1\}_{x \in L}$ and $\{w_x^2\}_{x \in L}$, such that $w_x^1, w_x^2 \in \mathbf{R}_L(x)$ for every $x \in L$, the following ensembles are computationally indistinguishable over $x \in L$ (resp.

statistically indistinguishable over $x \in L$, or perfectly identical):

$$\left\{ \text{View}_{V^*}(P(x, w_x^1), V^*(x, z)) \right\}_{x \in L, z \in \{0,1\}^*} \approx \left\{ \text{View}_{V^*}(P(x, w_x^2), V^*(x, z)) \right\}_{x \in L, z \in \{0,1\}^*}$$

Proofs and Arguments of Knowledge

Given a language $L \in \text{NP}$ and an instance x , a proof or argument of knowledge not only convinces the verifier that $x \in L$, but also to demonstrate that the prover possesses an NP-witness for x . This is formalized by the existence of an **extractor**: given black-box access to a machine that can successfully complete the proof or argument of knowledge on input x , the extractor can compute a witness for x .

Definition 16 (Proofs and arguments of knowledge [23, 8]). An interactive protocol $\Pi = (P, V)$ is a **proof of knowledge** (POK) (resp. **argument of knowledge**, AOK) of language L with respect to witness relation \mathbf{R}_L if Π is indeed an interactive proof (resp. argument) for L . Additionally, there exists a polynomial q , a negligible function ν , and a probabilistic oracle machine E , such that for every interactive machine P^* (resp. for every polynomially-sized machine P^*) and every $x \in L$, the following holds:

If $\Pr[\langle P^*, V \rangle(x) = 1] > \nu(|x|)$, then on input x and oracle access to $P^*(x)$, machine E outputs a string from the $\mathbf{R}_L(x)$ within an expected number of steps bounded by

$$\frac{q(|x|)}{\Pr[\langle P^*, V \rangle(x) = 1] - \nu(|x|)}$$

The machine E is called the knowledge extractor.

Special Soundness

Special-sound protocols are proofs of knowledge with a very rigid and useful structure.

Definition 17 (Special soundness [17]). A 4-round interactive proof (P, V) for language $L \in \text{NP}$ with witness relation \mathbf{R}_L is special sound (SS) with respect to \mathbf{R}_L if (P, V) is public-coin (i.e., verifier messages are segments of its random tape), and on input x , all verifier messages have length $g(|x|) \geq |x|$.

Moreover, there exists a deterministic polynomial-time extraction procedure X such that on input x , with all but negligible probability in $|x|$ over the choice of a uniform $\rho \in \{0, 1\}^{g(|x|)}$, for all $\alpha, \beta, \beta', \gamma, \gamma'$ such that $\beta \neq \beta'$, and $(\rho, \alpha, \beta, \gamma)$ and $(\rho, \alpha, \beta', \gamma')$ are both accepting transcripts of (P, V) on input x , $X(x, (\rho, \alpha, \beta, \gamma), (\rho, \alpha, \beta', \gamma'))$ outputs a witness $w \in \mathbf{R}_L(x)$.

1.2.5 Useful Known Constructions

The following protocols are useful in our various constructions:

- 4-round computational WI and SS proofs based on one-way functions. This can be instantiated with a parallel repetition of the Blum Hamiltonicity protocol [10] with 2-round statistically binding commitments constructed from one-way functions ([50, 35]).
- 4-round computational WI-AOK or $\text{poly}(n)$ -round statistical WI-AOK based on one-way functions. Again, this can be instantiated with the Blum Hamiltonicity protocol with the help of 2-round statistically binding commit-

ments (this actually gives a POK) from one-way functions, or statistically hiding commitments ([33]) from one-way functions.

- 4-round perfect WI-POK based on claw-free family of permutations. Again, this can be instantiated with the Blum Hamiltonicity protocol with the help of perfectly binding commitments constructed from claw-free family of permutations (see [24, Section 4.1.1]).

CHAPTER 2

CONCURRENT ZERO KNOWLEDGE, REVISITED

In the grand scheme of cryptography, concurrent security has been an active area of research, following the seminal works of [19, 23] from the early 90's. Yet, it is still not well-understood when and how concurrent security can be achieved. One potential reason for this might be the complexity of traditional analyses; In this chapter, we focus on generalizing and (in our eyes) simplifying the analysis of concurrent security in one of the most basic settings, namely that of *zero-knowledge protocols*.

It is no secret that cZK protocols are much harder to construct and analyze than standalone ZK protocols. Since their introduction by Dwork, Naor and Sahai [20], Richardson and Kilian [64] constructed the first concurrent ZK argument in the standard model without any extra set-up assumptions (the focus of this chapter). Their protocol, which uses a black-box simulator, requires $O(n^\epsilon)$ number of rounds. (See also the work of Canetti, Goldreich, Goldwasser and Micali [13] for a somewhat different and more detailed analysis of this protocol.) Kilian and Petrank [42] then introduced a new *oblivious* concurrent zero-knowledge simulator. Using this simulation technique they obtained a simpler and cleaner analysis, and additionally improved the round complexity of cZK protocols to $\tilde{O}(\log^2 n)$. Finally, the work of Prabhakaran, Rosen and Sahai [61] further simplifies and improves the analysis of the oblivious simulator, obtaining an the round complexity $\tilde{O}(\log n)$. This is essentially optimal given the work of Canetti, Kilian, Petrank and Rosen [14] (building on earlier works by [43] [65]), which shows that black-box cZK protocols for non-trivial languages require at least $\tilde{\Omega}(\log n)$ number of rounds.

Despite these simplifications and improvements, the analysis of concurrent zero-knowledge protocols remains quite complex. Furthermore, the different analyses are tailored to different types of protocols. In particular, the most refined analysis from [61] only seems to apply to *committed-verifier* protocols, where the verifier commits to its messages in advance; more specifically, as far as we know, the analysis has only been applied to generalizations of the Goldreich-Kahan ZK protocol [26]. For instance, no generalizations of the Feige-Shamir ZK protocol [23] have been analyzed using it; apart from theoretical interests, the Feige-Shamir ZK protocol is noteworthy due to its efficient instantiation via “sigma protocols” [17].

In this work, we focus on simplifying and generalizing current analysis techniques for concurrent ZK. More precisely, we provide a variant of Prabhakaran, Rosen and Sahai’s (PRS) analysis [61] of the Kilian-Petrank (KP) zero-knowledge simulator [42]. Our contribution is twofold:

- In our eyes, this analysis is simpler and more flexible than the original PRS analysis. In particular, the analysis also directly applies to more efficient variants of the KP-simulator, resulting in concurrent ZK protocols with “tight” [28, 24], and even “precise” [49] simulations (i.e., simulations where the running-time of the simulator is close to the running-time of the malicious verifier, in an execution-by-execution manner). We already achieve precision in a joint work with Pandey et. al. [51], but that requires a more elaborate analysis (building upon [61]). In Section 2.3, we show how to construct *precise* concurrent zero-knowledge arguments more easily using the analysis presented in this chapter.
- Our analysis applies to a broader range of protocols, and in particular to

“Feige-Shamir”-type protocols.¹ As a consequence, we establish a simple $\omega(\log n)$ -round concurrent zero-knowledge argument of knowledge for NP based on one-way functions. The same protocol construction also yields a $\text{poly}(n)$ -round concurrent statistical ZK argument of knowledge for NP, based on one-way functions (concurrent statistical ZK arguments were first constructed in [32] using a more complex protocol). Furthermore, a joint work with Lin et. al. [44] relies on our analysis to construct concurrent *non-malleable* zero-knowledge proofs for NP; our analysis is seemingly necessary for their proof, as their protocol is not of the committed-verifier type.

Previous techniques. Kilian and Petrank’s (KP) ingenious simulation technique relies on a static—and oblivious—rewinding schedule; namely, the simulator rewinds the adversarial verifier after some fixed number of messages, independent of the content of the messages and the interleaving schedule of the sessions. The crux of their analysis is to show that using this rewinding schedule, every session is “successfully rewound” at least once with high probability; in a successful rewind, the simulator can extract a “trapdoor” that will allow it to complete the simulation. To bound the failure probability, they rely on a subtle computation of conditional probabilities.

The elegant work of Prabhakaran, Rosen and Sahai (PRS) [61, 66, 62], on the other hand, directly analyze the probability space of the simulator, i.e., count the random tapes of the simulator; this makes the analysis both simpler and sharper. The idea is to show that each “bad” random tape (that produces a

¹It would seem that the analyses of [64] or [42] could also be applied to “Feige-Shamir”-type protocols, but this would result in protocols with sub-optimal round complexity.

failed simulation) can be mapped into super-polynomially many *distinct* “good” tapes. This is done by identifying random tape segments, called *rewinding intervals*, that can be “swapped” among each other in order to turn a bad tape into a good one². The crux of their proof is then to count how many such “swappings” actually generate new and distinct random tapes. However, complications arise because swappings performed on different rewinding intervals may overlap and even remove other possible rewinding intervals. Due to these issues, the PRS proof relies on a *global* analysis to lower-bound the overall number of available swappings. A bit more precisely, the PRS analysis focuses only on “disjoint” rewinding intervals, but performs an intricate computation based on the “multiplicity” on those intervals. A count with multiplicity is needed because the number of disjoint rewind intervals in general could not be guaranteed to be sufficiently large, at least in the case of $\omega(\log n)$ round protocols. (As we shall see, in our analysis, we are able to swap also non-disjoint rewinding intervals; as a result, we can avoid the count with multiplicities.)

Additionally, to enable this counting argument, the PRS analysis only bounds the failure probability of a “hybrid” simulator (which has access to the witnesses of input statements). To show that the real simulator is indistinguishable from the hybrid simulator, committed-verifier protocols are used; this is required to ensure that when changing the hybrid simulator (which uses the actual witness) to the real simulator (which doesn’t know the actual witness), indistinguishability holds despite the rewinds performed by the simulator. Intuitively, the committed-verifier property ensures that the rewinds are “harmless”.

²Here we use the terminologies from Rosen’s thesis [66].

Our techniques. We show how to directly analyze the failure probability of the *actual* simulator (as opposed to a hybrid one), while (in our eyes) simplifying the counting argument. Our key step is to identify a *stronger* notion of rewinding intervals, which we call *composable blocks*. Just like rewinding intervals, properties of composable blocks guarantee that a “swap” will generate a new good random tape; moreover, these same properties are closed under *composition*—namely the swapping of one such block leaves other composable blocks intact, even if these composable blocks are not disjoint. By this new composition property, it is enough to identify K composable blocks to conclude that the simulation fails with probability less than 2^{-K} .

In essence, our proof will consist of two simple steps: First, we establish *local* properties of a composable block (namely that a swap generates one new good random tape, and that swappings are composable); then, we count the number of composable blocks on a bad random tape; as we shall see, each round in the protocol gives rise to a new composable block. As such, our analysis conveys a strong intuition of how “each additional round of the protocol halves the simulator’s failing probability”. However, we emphasize that our techniques do not improve the “quantity” of the count (e.g., does not improve upon the round-complexity of the PRS protocol).

To employ this new notion of composable blocks, we consider and analyze a “lazy” variant of the KP simulator. Intuitively, the lazy KP simulator is identical to the KP simulator but only makes use of information gathered in its rewinds after some delay. The lazy KP simulator can only fail more often than the original KP simulator (and thus our analysis indirectly also applies to the KP simulator); yet, considering this “weaker” simulator enables our way of directly

analyzing the failure probability of the simulation. In a sense, much like making a stronger inductive hypothesis can simplify the inductive step, our stronger notion of **composable** blocks and our weaker lazy KP simulator enable and simplify the analysis.

After directly bounding the failure probability of the real simulator, we provide a simple hybrid argument to show that the output of the simulator is indistinguishable from the view of the verifier. The base case of this hybrid argument considers only a “straight-line” (i.e., a non-rewinding) execution, and as such the analysis is not limited to committed-verifier protocols.

Overview. In this chapter we re-prove the following theorem (first shown in [61, 32]).

Theorem 1. *Assume the existence of one-way functions. Then every language in NP has a $\omega(\log n)$ -round concurrent black-box ZK argument of knowledge, and a $\text{poly}(n)$ -round concurrent black-box statistical-ZK argument of knowledge.*

We describe our protocol in Section 2.1, and our simulator in Section 2.2. Using the same analysis, we discuss how to achieve precise cZK in Section 2.3. For completeness, we also provide a brief comparison with the PRS [61] analysis in Section 2.4.

2.1 The Protocol

Our concurrent ZK protocol **ConcZKArg** (also used in [59]) is a slight variant of the precise ZK protocol of [49], which in turn is a generalization of the Feige-

Shamir protocol [23]. The protocol for language L proceeds in three stages, given a security parameter n , a common input statement $x \in \{0, 1\}^n$, and a “round-parameter” $k \in \omega(\log n)$:

Stage Init: The verifier picks two random strings $r_1, r_2 \in \{0, 1\}^n$ and sends their images $c_1 = f(r_1)$, $c_2 = f(r_2)$ through a one-way function f to the prover. The verifier then provides, in parallel, k instances of a 4-round computationally-WI and SS proof of knowledge of the NP statement “ c_1 or c_2 is in the image set of f ” (a witness here would be a pre-image of c_1 or c_2). The first two (out of four) messages of each SS-POK are exchanged in this stage. The end of Stage Init is called the **start** of the protocol.

Stage 1: k message exchanges occur in Stage 1. In the j^{th} iteration, the prover sends $\beta_j \in \{0, 1\}^n$, a random second last message of the j^{th} SS-POK, and the verifier replies with the last message γ_j of the SS-POK. These k iterations are called *slots*. A slot is *convincing* if the verifier produces an accepting proof. If there is ever an *unconvincing* slot, the prover aborts the whole session. The end of Stage 1 (after k convincing slots) is called the **end** of the protocol.

Stage 2: The prover provides a 4-round computational-WI (resp. $\text{poly}(n)$ -round statistical-WI) argument of knowledge of the statement “ $x \in L$, or one of c_1 or c_2 is in the image set of f ”.

Completeness and soundness/proof of knowledge follows directly from the proof of Feige and Shamir [23]; in fact, the protocol is an instantiation of theirs. Intuitively, to cheat in the protocol a prover must “know” an inverse to c_1 or c_2 (since Stage 2 is an argument of knowledge), which requires inverting the

Protocol ConcZKArg:

Common Input: an instance x of a language L with witness relation \mathbf{R}_L .

Auxiliary Input for Prover: a witness w , such that $(x, w) \in \mathbf{R}_L(x)$.

Stage Init:

V uniformly chooses $r_1, r_2 \in \{0, 1\}^n$.

$V \rightarrow P$: $c_1 = f(r_1), c_2 = f(r_2)$ for a one-way function f .

$V \leftrightarrow P$: Exchange in parallel (interactively) the first two messages $\vec{\alpha}_1, \dots, \vec{\alpha}_k$ of k copies of 4-round computational-WI and SS proofs on common input (c_1, c_2) with respect to the witness relation:

$$\mathbf{R}_f(c_1, c_2) = \{r : f(r) = c_1 \text{ or } f(r) = c_2\}$$

Note that V acts as the prover in these SS-POK's.

We say the protocol has reached **start** (of Stage 1) if all messages in Stage Init are exchanged.

Stage 1: For $j = 1$ to k do the following (called a *slot*)

$P \rightarrow V$: The second last message $\beta_j \leftarrow \{0, 1\}^n$ of the j^{th} SS-POK.

$V \rightarrow P$: The last message γ_j of the j^{th} SS-POK.

We say the protocol has reached **end** (of Stage 1) if all k SS-POK are accepted.

Stage 2:

$P \leftrightarrow V$: a 4-round computational-WI (or $\text{poly}(n)$ -round statistical WI) argument of knowledge from P to V on common input (c_1, c_2, x) with respect to the witness relation:

$$\mathbf{R}_{f \vee L}(c_1, c_2, x) = \{(r, w) : r \in \mathbf{R}_f(c_1, c_2) \text{ or } w \in \mathbf{R}_L(x)\}$$

Figure 2.1: Concurrent ZK argument of knowledge for NP with round parameter k .

one-way function f . A formal description of protocol ConcZKArg is shown in Figure 2.1.

2.2 The “Lazy KP” ZK simulator

We show that whenever k is super logarithmic (i.e. $k = \omega(\log n)$), our protocol is concurrent ZK. This requires us to construct a simulator $\text{Sim} = \text{Sim}^{V^*(x,z)}(x)$ that given input instance $x \in L$ and black-box access to $V^*(x, z)$, outputs a view that is indistinguishable from the real view of $V^*(x, z)$. On a very high-level, the simulation follows that of Richardson and Kilian [64]. The simulator simulates Stage Init and Stage 1 of the protocol by following the honest prover strategy, and attempts to rewind one of the *slots* (i.e. the last two messages of the special-sound proofs provided by V^*). If the simulator obtains two *matching* convincing slots, i.e., the slots are from the same round of the protocol and share the same initial transcript, the special-soundness property allows the simulator to compute a **fake** witness r such that $f(r) = c_1$ or c_2 . This fake witness can then be used to simulate Stage 2 of the protocol. Towards this goal, we let Sim be an *oblivious* black-box simulator similar to [42].

Description of Sim. Let n be the security parameter, m be a bound on the number of concurrent sessions invoked by V^* and T be the total number of messages exchanged, bounded by $O(mk)$, a polynomial in n . Keep in mind that during black-box simulation, we assume without loss of generality that V^* is deterministic; therefore the view of V^* is just the transcript of its interaction with the honest prover.

In order to extract a fake witness from V^* , Sim follows an oblivious rewinding schedule based only on the number of messages exchanged so far, just like in [42] and [61]. During the oblivious simulation, Sim keeps a repository of all messages generated by V^* among all rewinds; whenever Sim encounters Stage

2 of the protocol, Sim looks for matching convincing slots in this repository to compute the required fake witness. More precisely, Sim uses the recursive procedure `lazy-rewind` described below.

At a high level, $\text{lazy-rewind}(t, \mathcal{V}, \mathcal{T}) \rightarrow (\mathcal{V}', \mathcal{T}')$ attempts to recursively simulate $V^*(x, z)$ for t messages starting from a partial view \mathcal{V} of V^* , with the help of a repository of messages generated by V^* during rewinds, \mathcal{T} (formally just a set of all simulator query and verifier message pairs). If `lazy-rewind` is successful, it outputs a longer view \mathcal{V}' of V^* (that contains exactly t more verifier messages than \mathcal{V}), and an updated repository \mathcal{T}' including verifier messages that `lazy-rewind` gathered from various rewinds (and most likely contains more verifier messages than what is recorded in \mathcal{V}'). Sim simply outputs the view produced by $\text{lazy-rewind}(T, \mathcal{V} = \emptyset, \mathcal{T} = \emptyset)$, i.e., `lazy-rewind` starting from the empty view and an empty repository.

Description of `lazy-rewind`(t, s, h). At the base case of the recursion ($t = 1$), `lazy-rewind` receives a message from V^* and produces a prover response. `lazy-rewind` behaves identically to an honest prover to generate Stage Init and Stage 1 messages. Whenever a session reaches end, `lazy-rewind` will attempt to compute a fake witness r for the session ($f(r) = c_1$ or c_2) by searching \mathcal{T} for matching convincing slots. If this is successful, the fake witness r is used to generate prover messages in Stage 2 of this session (i.e. the WI-POK). Otherwise, `lazy-rewind` outputs \perp , which in turn causes Sim to output \perp as well.³ In the end, `lazy-rewind` outputs the updated view \mathcal{V}' of V^* (the input view appended with the newly exchanged pair of messages), and the updated repository \mathcal{T}' (the input repository

³We distinguish between legitimate failures, i.e., Sim may abort just like a prover should V^* fail to follow the protocol, and simulation failures, i.e., Sim outputs \perp if it fails to compute a fake witness r .

inserted with the newly exchanged pair of messages).

When $t > 1$, $\text{lazy-rewind}(t, \mathcal{V}, \mathcal{T})$ proceeds roughly as follows: It first recursively simulates V^* for $t/2$ messages twice starting from the partial view \mathcal{V} . Then, continuing from one of those simulations, lazy-rewind recursively simulates V^* for another $t/2$ messages, twice. More formally, $\text{lazy-rewind}(t, \mathcal{V}, \mathcal{T})$ calls itself four times as follows:

1. $(\mathcal{V}_1, \mathcal{T}_1) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$.
2. $(\mathcal{V}_2, \mathcal{T}_2) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$. Merge \mathcal{T}_1 and \mathcal{T}_2 into a larger repository of messages \mathcal{T}' .
3. $(\mathcal{V}_3, \mathcal{T}_3) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}_1, \mathcal{T}')$.
4. $(\mathcal{V}_4, \mathcal{T}_4) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}_1, \mathcal{T}')$. Merge \mathcal{T}_3 and \mathcal{T}_4 into a larger repository of messages \mathcal{T}'' .
5. Output $(\mathcal{V}_3, \mathcal{T}'')$.

Because the first two recursive calls to lazy-rewind (respectively the last two calls) have identical inputs (they differ only because they use different segments of Sim 's random tape), they are called *sibling* calls. See Figure 2.2 for an illustration of the rewinding schedule, and Figure 2.3 for a pseudo-code description.

Let us describe some terminology that is useful for the analysis Sim and lazy-rewind . Because Sim follows an oblivious rewinding schedule, it always makes a fixed set of calls to lazy-rewind at fixed moments in the simulation, and it always “connects” these calls of lazy-rewind in a fixed way to generate partial views of V^* . Intuitively, a *thread* is one of these fixed connections.

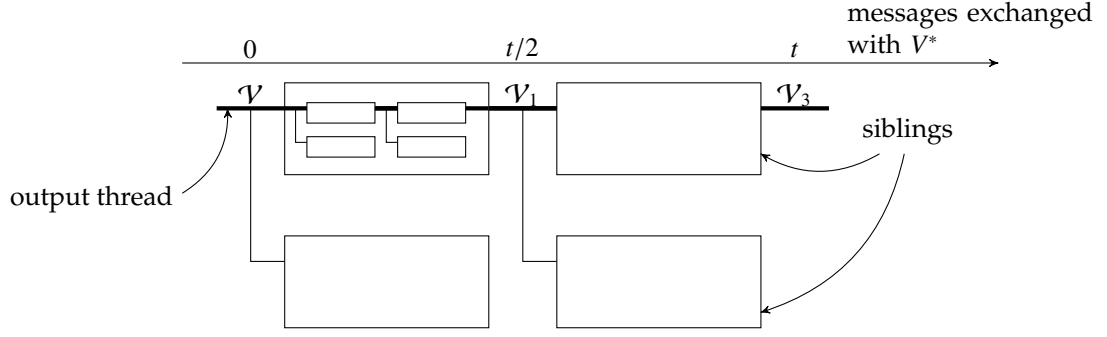


Figure 2.2: A pictorial representation of the rewinding schedule of lazy-rewind. The boxes represent *blocks*, and the lines represent *threads*. If this is the top level call (i.e., $\text{lazy-rewind}(T, \emptyset, \emptyset)$), then the thicker thread is the *output thread*, whose view is the output of Sim.

Definition 18 (Threads). A *thread* is a sequence of 0's and 1's; from the beginning of the simulation, this sequence specifies, whenever a pair of sibling calls are encountered, whether to follow the first or second sibling call of lazy-rewind, respectively. (A sequence may terminate prematurely to specify a “partial” thread.) The thread $00 \cdots 0$ (of sufficient length) is the thread that follows the first sibling calls to the end of the simulation, and is called the *output thread* because the view of V^* generated on this thread is the output of Sim.

Given an execution of Sim (on an input $x \in L$ and a random tape), a *block* intuitively refers to the “location” (in the static rewinding schedule) of a call to lazy-rewind, as well as the actual simulation performed by the call.

Definition 19 (Blocks). Given an execution of Sim, a **block** B is a pair $B = (B_{\text{loc}}, B_{\text{content}})$, where B_{loc} specifies the location of a call of lazy-rewind and B_{content} specifies the inputs and randomness of the same call. Formally B_{loc} is a partial thread (that leads to and includes the lazy-rewind call), and B_{content} is just the inputs and random tape used by the lazy-rewind call, i.e., $(t, \mathcal{V}, \mathcal{T}, r)$. We say a block C is **contained** in block B if the recursive call of lazy-rewind correspond-

lazy-rewind($t, \mathcal{V}, \mathcal{T}$):

1. **Base Case:** $t = 1$. Exchange one pair of messages with V^* .

- (a) If the next scheduled message is from an aborted session where V^* has deviated from the protocol (e.g., there has been an unconvincing slot), return $(\mathcal{V} \parallel \text{abort}, \mathcal{T})$ (i.e. do nothing).
- (b) If the next scheduled message is a Stage Init or Stage 1 prover message for session i , compute a message p following the honest prover strategy. Let v be the response of V^* ; if v deviates from the protocol (e.g., v is an unconvincing last message of a SS-POK) abort session i .
- (c) If the next scheduled message is a Stage 2 prover message for session i , use the computed fake witness to compute the prover messages p for the WI-AOK, and let v be the verifier response. Note that a fake witness must have already been computed to reach this point in the simulation; see next bullet.
- (d) After exchanging a pair of messages p and v , if we reach the end of a session, attempt to compute a fake witness of the session using the special-soundness property and previous messages stored in the repository \mathcal{T} (in particular are looking for matching convincing slots for session i). If lazy-rewind fails to compute a fake witness, output \perp .
- (e) Output $(\mathcal{V} \parallel p \parallel v, \mathcal{T} \cup \{\mathcal{V} \parallel p \parallel v\})$, i.e., extend the input partial view with the message pair (p, v) and enlarge the input repository with the new message generated by V^* .

2. **Recursive step**

Simulate the first $t/2$ messages twice

(a) $(\mathcal{V}_1, \mathcal{T}_1) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$

(b) $(\mathcal{V}_2, \mathcal{T}_2) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$

Simulate the second $t/2$ messages twice

(c) $(\mathcal{V}_3, \mathcal{T}_3) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T}_1 \cup \mathcal{T}_2)$

(d) $(\mathcal{V}_4, \mathcal{T}_4) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T}_1 \cup \mathcal{T}_2)$

(e) output $(\mathcal{V}_3, h_3 \cup h_4)$

Figure 2.3: The recursive procedure used by Sim—the “lazy” KP simulator.

ing to block C is nested inside the recursive call of *lazy-rewind* corresponding to block B .

Due to the recursive nature of *lazy-rewind*, every block would contain four “smaller” blocks; of these four blocks, we call the first pair (respectively the second pair) **sibling blocks**, as they correspond to sibling calls of *lazy-rewind*. Finally, we say a block contains a thread if the thread “passes through” the block.

Definition 20 (Threads in a Block). Given an execution of *Sim*, we say a block B contains a thread h if B_{loc} is a prefix of h .

Since *lazy-rewind* does not update the message repository \mathcal{T} between sibling recursive calls (sibling blocks) we call it *lazy*. This departs from previous works such as [42, 61]⁴. We have also changed how blocks are threaded together from [42, 61]. In *lazy-rewind*, the second pair of recursive calls are continued from the first recursive call of the first pair (i.e. continued from view \mathcal{V}_1). This is similar to the precise simulation of [49] and [51] (and will be useful later when we discuss precision in Section 2.3); [42] and [61], in contrast, continue the recursive calls from the view \mathcal{V}_2 . Due to the new symmetry introduced by lazy simulation, either choice will work with our analysis. See Figure 2.2 for an illustration of blocks, threads and siblings in an execution of *lazy-rewind*.

2.2.1 Proof Overview

In order to prove the correctness of the simulation, we need to show that for every adversarial verifier V^* , the simulator runs in polynomial time and the

⁴This extra symmetry is part of what enables us to analyze the *Sim* directly (as opposed to a “hybrid” simulator as in [61]); see Section 2.4 for a comparison.

output distribution is “correct”. The running time of Sim can be bound just as in [42, 61]. Sim spends a maximum of $\text{poly}(n)$ time on responding to each verifier message. It follows from the recursive structure of the simulator that the number of messages exchanged is doubled for each level of the recursion; since we have a recursive depth of $\log_2 T$, the running time of the simulator is bounded by $\text{poly}(n) \cdot T \cdot 2^{\log_2 T} = \text{poly}(n) \cdot T^2 = \text{poly}(n)$.

Intuitively, the correctness of the output view follows from the fact that Sim chooses Stage Init and Stage 1 messages honestly, and that the protocol used in Stage 2 is witness indistinguishable (this requires a proof later since Sim performs rewinds). Therefore, as long as Sim gets stuck (outputs \perp) with negligible probability, taken over the random tapes of Sim (the random tape of V^* is fixed during black box simulation), the output distribution is correct. Towards this goal we will show that the probability of getting stuck at any point in the simulation is negligible.

Recall that Sim can only get stuck on a particular thread when the simulation reaches the end of some session and could not extract the fake witness. Following the approach of [61], we show that the probability of getting stuck on *any session* and *any thread* is negligible. Since there are only polynomially many sessions and threads, the main theorem follows by the union bound.

Fix any thread h and session i ; from now on we refer to it as the “main” thread and the “main” session, and call all other threads and sessions “auxiliary”. We say a random tape of Sim is *bad* if Sim gets stuck at the end of main session i on the main thread thread h ; all other random tapes are called *good* (including those that got stuck on an auxiliary session or thread). The high-level idea, just like in [61], is to show that for every *bad* random tape, there exists

super-polynomially many **good** random tapes. Furthermore, the **good** tapes corresponding to any two **bad** tapes are disjoint. Hence the probability of a tape being **bad** is negligible. From here on, **start** and **end** refer to those on the main session and thread unless otherwise noted.

Here is how we generate **good** random tapes from **bad** ones. Recall that on a **bad** tape, the simulator reaches **end** without extracting a “fake witness”. Hence, all slots on the main thread are convincing (or else we would never reach **end**), but no corresponding convincing slots are on an auxiliary thread prior to **end** (since otherwise Sim would have extracted a witness). Intuitively, to generate a **good** tape from a **bad** one we just need to “swap” a convincing slot from the main thread into an auxiliary thread. After the swapping, should the simulation reaches **end** of the main session on the main thread, the newly formed convincing slot on the auxiliary thread, together with the corresponding convincing slot on the main thread, will allow Sim to compute the fake witness. Hence the simulation may continue on without getting stuck. So far we have not deviated from the analysis of [61].

To actually “swap” convincing slots, we modify the random tape of Sim. The basic operation that we perform on the random tape is to exchange the randomness used by sibling blocks (i.e., the segments of the random tape used to simulate these blocks). Since sibling blocks are identical modulo randomness, swapping the random tape between siblings swaps the simulation result in the two blocks *exactly*. (In the rest of the paper, we use the convention that after swapping a block B with its sibling B' , the “new block B ” refers to the block in the old location of B' with the same content as the “old block B ”, i.e., $(B'_{\text{loc}}, B_{\text{content}})$.) Note that this “exact swap” property is made possible by the lazy

nature of Sim; the same property does not hold for the KP simulator where the second sibling benefits from fake witnesses extracted during the execution of the first sibling.

Intuitively, we call a block on the main thread **composable** if it satisfies the following properties:

Goodness. Swapping a composable block with its sibling produces a good random tape.

Composability. The above swap leaves other composable blocks on the main thread composable.

Reversibility. Given the random tape obtained after swapping a composable block, there is a procedure **undo** that reverses the swap. This ensures that the resulting good tape is unique.

Consider K composable blocks with an ordering such that each swap will leave the successive composable blocks still composable. Then, we can generate $2^K - 1$ good random tapes by choosing to swap each block or not in the ordering. By a simple counting argument, we will show that for any bad tape, there are $k - 2 \log_2 T$ composable blocks with an ordering, therefore generating $2^{k-2 \log_2 T}$ distinct good tapes. We then use the **undo** procedure to show that different bad tapes generate different good tapes. Thus, if $k \in \omega(\log_2 T) = \omega(\log n)$, the probability of having a bad tape is negligible.

2.2.2 The Actual Proof

Formally, Sim may output \perp for two reasons. Firstly, it may reach **end** without encountering two matching convincing slots after the **start** of the session; we call this a *rewinding failure*. Secondly, Sim may not be able to compute a fake witness even though it has access to matching special-sound transcripts; we call this a *special-sound failure*. Special-sound failures are easy to upper bound; see Claim 8. As mentioned, the main part of the proof is bounding the probability of rewinding failures.

Composable Blocks

We first define the notion of **composable blocks** and show that they satisfy the three properties of *goodness*, *composability* and *reversibility*. Let us fix a particular main session and main thread, and formally define a random tape to be **bad** if Sim encounters a *rewinding failure* in the main session on the main thread; otherwise a random tape is **good**. From here on **start** and **end** refers to those of the main session and main thread, unless otherwise noted.

Definition 21 (Composable Block). Consider an execution of Sim with any random tape (not necessarily **bad**). A block B , with sibling B' , is called a **composable block** (with respect to the main thread and session), if it satisfies the following conditions:

Main block condition: B contains the main thread h , a convincing slot of the main session (not necessarily on the main thread) and does not contain **start** (of the main session on the main thread). The last condition is equivalent to saying that the prefix of B contains **start**.

Sibling condition: B' does not contain any `end` (of the main session on the main thread).

Tracing condition: The simulation after `start` but before B contains only convincing slots on the main thread h , and contains no convincing slots on the auxiliary threads.

As we will soon see, the Main block condition and the Sibling condition implies goodness and composability, while the Tracing condition enables the undo procedure, which implies reversibility. We also define an ordering relation $>$ on composable blocks.

Definition 22. Let C and B be two blocks on a common thread. We write $C > B$ iff

- C and B are disjoint, and C occurs before B (Case 1 in Figure 2.4), or
- C and B are not disjoint, and C is a larger block that contains B (Case 2 in Figure 2.4)

Note that given two blocks on the same thread, if they are not disjoint, then one must contain another. Thus $>$ is a total order on any set of blocks that share a common thread.

Finally, we define a deterministic `undo` function on random tapes in order to achieve reversibility:

- Given a random tape τ' , execute `lazy-rewind` with the tape τ' . Call a block that does not contain the main thread *special* if it contains a convincing slot of the main session.

- Let D be the first special block after **start**; that is, any other special block E after **start** satisfies $D > E$. Swap the parts of τ' used by D and its sibling, and output the new random tape.

Claim 2. *Let τ be a random tape (not necessarily **bad**). Let B be a **composable** block with sibling B' when **lazy-rewind** is executed with random tape τ , and let \mathcal{V} be the common prefix of B and B' . Furthermore, let τ' be the random tape obtained after swapping the blocks B and B' . Then:*

1. [Goodness]: τ' is a **good** random tape.
2. [Composability]: Any **composable** block C on τ with $C > B$ is still **composable** on τ' .
3. [Reversibility]: $\text{undo}(\tau') = \tau$.

Proof. Recall that after the swapping, blocks B and B' are exchanged in the simulation.

Goodness When **lazy-rewind** is executed with τ' , B' will now be on the main thread (see Figure 2.4). Recall that B' does not contain any **end** of the main session (sibling condition). Thus, if the **end** of the main session ever occurs on the main thread, it will occur after both B and B' are executed. In that case, both the convincing slot in B (which is now in an auxiliary thread) and the corresponding convincing slot on the main thread (which must be there before **end** occurs) together forms a matching pair of convincing slots that occurs after **start**.

Moreover, this pair of convincing slots occur before **end**. Thus τ' is a **good** tape.

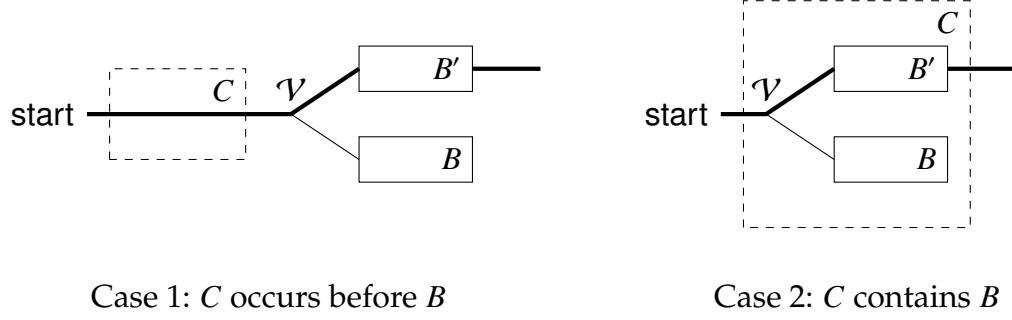


Figure 2.4: Two possible block diagrams after the swapping procedure in Claim 2 (B and B' is swapped). The main thread is shown in a thick line, and a composable block $C > B$, drawn with dashed lines, is shown in two possible configurations.

Composability Given a composable block $C > B$ with sibling C' on τ , we have two cases as shown in Figure 2.4. In case 1, when C is disjoint from B , the swapping of B and B' does not change the simulation inside C , C' , and between start and C . Respectively, this leaves the main block condition, sibling condition, and tracing condition of C intact on τ' . On the other hand, in case 2 where C contains B , the swapping of B and B' again leaves the simulation inside C' (not shown) and between start and C unchanged, keeping the sibling condition and tracing condition intact. In addition, since C still contains B under τ' , and B in turn contains a convincing slot, the main block condition still holds as well (other parts of C may have changed). In both cases, C continues to be a composable block on τ' .

Reversibility Finally, we need to show that $\text{undo}(\tau') = \tau$. After the swap (executing with random tape τ'), block B no longer contains the main thread and contains a convincing slot; it is therefore a special block. Next we show that any block $C > B$ is not special. Either C occurs strictly before B , or it contains B (in this case C also contains B'). In the first case, block C is unchanged during the swap, and therefore is not special because it

does not contain a convincing slot (tracing condition). In the second case, since C contains B' and therefore the main thread, it is not special. Thus, undo will always locate B as the first special block and perform the correct inverse swapping to recover τ .⁵ \square

The next claim demonstrates how to compose multiple composable blocks.

Claim 3. *Let τ be a **bad** random tape, $\mathcal{B} = \{B_1, \dots, B_p\}$ be a set of **composable** blocks for τ . Then, we can generate a set of **good** random tapes, $S(\tau, \mathcal{B})$, by swapping the various **composable** blocks in \mathcal{B} , so that the following holds:*

1. $|S(\tau, \mathcal{B})| \geq 2^p - 1$.
2. For any bad tape $\tau' \neq \tau$ and any set of **composable** blocks \mathcal{B}' for τ' , $S(\tau, \mathcal{B}) \cap S(\tau', \mathcal{B}') = \emptyset$.

Proof. Since all composable blocks lie on the main thread, there is a total ordering of the blocks. Without loss of generality, let $B_1 > B_2 \dots > B_p$. Consider any non-empty subsequence of $1, \dots, p$, say u_1, \dots, u_q . There are $2^p - 1$ such sequences. Let $\tau_{u_1 \dots u_q}$ be the random tapes obtained from τ by swapping the blocks B_{u_i} with its sibling, in the order of $i = q, q-1, \dots, 1$.

From Claim 2, it follows that $\tau_{u_1 \dots u_q}$ is a good random tape. We further note that given $\tau_{u_1 \dots u_q}$, we can recover the blocks B_{u_1}, \dots, B_{u_q} by repeatedly applying undo until we reach a bad tape (it will always be τ). Therefore given two different subsequences, u_1, \dots, u_q and $v_1, \dots, v_{q'}$, we must have $\tau_{u_1 \dots u_q} \neq \tau_{v_1 \dots v_{q'}}$ in order

⁵ Note that we here rely on the “exact” swapping of sibling blocks (a consequence of the lazy property of Sim). Suppose that sibling blocks are not symmetric and that the second sibling uses information obtained in the first sibling to compute fake witnesses. Then, if the end of an auxiliary session occurs before the convincing slot in B' , it may now output \perp after the swapping (since it has lost the information collected in B after the swap). In this case, block B would not exist when executing Sim with random tape τ' , and undo would fail.

for undo to recover a different set of swapped blocks. Thus, we obtain $2^p - 1$ distinct good random tapes.

Similarly, take any $\alpha \in S(\tau, \mathcal{B})$ and $\beta \in S(\tau', \mathcal{B}')$ (good tapes produced by swapping from τ and τ' , respectively). Applying undo repeatedly on α until the result is a bad tape will result in τ , while applying the same procedure on β will give τ' . If $\tau \neq \tau'$, then we must have $\alpha \neq \beta$. \square

Corollary 4. *Suppose every bad random tape has p composable blocks. Then, the probability of a random tape being bad is at most $1/2^p$*

Number of Composable Blocks

We now proceed to count the number of composable blocks. First we introduce the notion of *minimal containing blocks* (this is identical to minimal rewinding intervals as defined by [61]). For each slot, its *minimal containing block* is the minimal block on the main thread that contains the slot. Claims 5 and 6 below together show that there are at least $k - 2 \log T$ composable blocks when we run Sim with a bad tape. Claim 5, which counts the number of minimal containing blocks, is identical to [61]; we include it here for completeness.

Claim 5. *In an execution of Sim with a bad random tape, there are at least k minimal containing blocks.*

Proof. As observed earlier, on a bad tape there will be k convincing slots of the main session on the main thread (in order to reach end). We merely need to show that for each slot, its respective minimal containing blocks are distinct. Suppose that two slots share the same minimal containing block of length t . Since slots on the same thread are disjoint, we reach a contradiction as one of

the slots must be properly contained in one of the two smaller blocks of size $t/2$. \square

Claim 6. *Consider an execution of **Sim** with a **bad** random tape τ . If there are k' minimal containing block, then there are at least $k' - 2 \log T$ **composable** blocks.*

Proof. Let B be a minimal containing block that does not contain **start** or **end**. Since **start** (or **end**) can only be in at most $\log T$ different blocks on the main thread (since that is the recursion depth), we conclude that there are at least $k' - 2 \log T$ such blocks.⁶ It remains to show that B is a **composable** block. Let B' be the sibling of B .

The main block condition of **composable** blocks follows directly, while the tracing condition on the main thread actually holds for the whole simulation from **start** to **end**, since τ is a **bad** random tape. Thus, we only need to show that the sibling condition is satisfied, i.e. B' does not contain **end**. Assume to the contrary that B' does contain **end**. Since B and B' are siblings with a common starting point and B contains a slot of the main session, B' must contain that same slot in a convincing manner in order to reach **end**. On the other hand, B does not contain **end**. Thus B' will be executed before the main thread reaches **end** (if at all), and this convincing slot will allow **Sim** to compute the witness of the main session by the same argument in Claim 2. This contradicts the fact that τ is a **bad** tape. \square

⁶This is the same counting argument used in [61] to count minimal rewinding intervals without **start** or **end**.

Concluding the Proof

We first show that Sim gets stuck with negligible probability, and then use it in Claim 9 to conclude that the output distribution of Sim^{V^*} is computationally (resp. statistically) indistinguishable from the real view of V^* .

Claim 7. *Sim encounters rewinding failures with negligible probability.*

Proof. As mentioned before, since there are only polynomially many sessions and threads, it suffices to show that the probability of the simulator getting stuck on any fixed thread and session is negligible. The union bound then shows that Sim overall gets stuck with negligible probability

For any fixed thread and session, combining Claim 3, 5 and 6 shows that a random tape is **bad** with probability at most

$$\frac{1}{2^{k-2 \log T}}$$

This is negligible in n since T is polynomial in n and $k = \omega(\log n)$. \square

Claim 8. *Sim encounters special-sound failures with negligible probability.*

Proof. Suppose for the sake of contradiction that Sim encounters special-sound failures with non-negligible probability. Consider an unbounded adversarial prover P^* that forwards the prover messages of the prefix of the SS-POK (Stage Init), in a random session and random thread from an execution of Sim^{V^*} , to an outside honest verifier V_{SS} of the SS-POK (essentially we are forwarding messages between V^* , who acts as the prover of the SS-POK in Stage Init, to the outside honest verifier V_{SS}). Since an execution of Sim^{V^*} only has polynomially

many instances of Stage Init, P^* would contradict the special-soundness property with non-negligible probability (i.e., produce a prefix of the SS-POK where it is possible for the witness-computing procedure to fail, even when supplied with two different completions of the prefix).

The actual “forwarding” procedure of P^* has a subtlety due to the rewinding nature of Sim. In the middle of forwarding the prefix of a SS-POK from V^* to V_{SS} , Sim may decide to rewind V^* partially to an earlier point in the SS-POK proof. In a naive forwarding scheme, this would require V_{SS} to be rewound as well to generate fresh verifier messages (which cannot be done). Fortunately, since the prefix of our SS protocol has only 2 messages (non-interactive), such a rewinding cannot occur. \square

Claim 9. *If the argument of knowledge in Stage 2 is WI (resp. statistical WI), then the ensembles $\{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*}$ and $\{S^{V^*(x,z)}(x)\}_{x \in L, z \in \{0,1\}^*}$ are computationally (resp. statistically) indistinguishable over $x \in L$.*

Proof. We consider polynomially many intermediate hybrids Sim_i , $0 \leq i \leq m+1$, that receive the real witnesses to the statements x_1, \dots, x_m . Sim_i proceeds as Sim until the i^{th} Stage 2 proof on the output thread, after which Sim_i continues in a straight-line simulation with V^* using the real witnesses for Stage 2 proofs. Sim_i will output \perp , however, should Sim encounter a rewinding or special-sound failure during the i^{th} proof. Clearly, Sim_0 generates $\text{View}_{V^*}(P(x), V^*(x, z))$ and Sim_{m+1} generates $S^{V^*(x,z)}(x)$. Thus, it is enough to show that for all i , the output of Sim_i and Sim_{i+1} , are computationally (resp. statistically) indistinguishable.

We introduce yet another hybrid Sim'_i that proceeds as Sim_i except that it utilizes the extracted fake witness for the i^{th} proof. Sim'_i and Sim_{i+1} differ only

in that Sim_{i+1} has more chances to output \perp . But by Claim 7 and 8, both actually outputs \perp with negligible probability, and therefore they are statistically close. On the other hand, Sim_i and Sim'_i differ only in the i^{th} proof, which both simulators run in a straight-line fashion, without rewinds. Therefore they are computationally (resp. statistically) indistinguishable by the WI property of the Stage 2 proof. Thus, the output of Sim_i and Sim_{i+1} are indeed computationally (resp. statistically) indistinguishable. \square

Claim 9 completes the proof of Theorem 1.

Remark. Since we have shown that our lazy simulator is a concurrent zero-knowledge simulator, it follows directly that the KP simulator is also a concurrent zero-knowledge simulator: because the KP simulator receives more information than the lazy simulator at any point during the simulation (i.e. a bigger history repository h), the probability that the KP simulator outputs \perp is no more than the probability that the lazy simulator outputs \perp . Thus, the same argument presented in Claim 9 can be applied also to the KP simulator.

2.2.3 Improving the Running Time of the Simulator

A faster simulator gives tighter “knowledge security” [28, 24]. In this section, we bound the running time Sim by bounding the number of queries the Sim makes to V^* . Recall that T is the maximum number of queries needed in a straight-line execution of V^* . This means the recursive depth of $\text{lazy-rewind}(T, \cdot, \cdot)$ invoked by Sim is at most $\log_2 T$. Since lazy-rewind doubles the number of queries per recursive depth, Sim makes at most $T2^{\log_2 T} = T^2$ queries.

We can reduce the number of queries by following the approach of [51]. Currently, when *lazy-rewind* is asked to simulate t messages (a block of size t), it divides the t messages into two halves and recursively calls itself on each half (creating blocks of size $t/2$). One approach would be for *lazy-rewind* to divide each block into smaller sub-blocks, thus reducing the recursive depth. For example, an illustration of dividing a block into 3 equal parts is shown in Figure 2.5 (this means $\text{lazy-rewind}(t, \cdot, \cdot)$ would make 6 recursive calls of the form $\text{lazy-rewind}(t/3, \cdot, \cdot)$). Suppose we divide each block into g equal sized sub-blocks; we call g the *splitting factor*. Then it immediately follows that the recursive depth of $\text{lazy-rewind}(T, \cdot, \cdot)$ becomes $\log_g T$, and the number of queries made by *Sim* is reduced to at most $T2^{\log_g T}$.

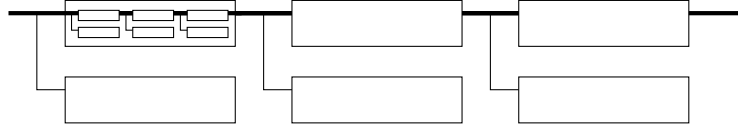


Figure 2.5: A pictorial representation of a rewinding schedule with splitting factor 3.

Now that the *Sim* is making less queries overall, can it still successfully generate a view of V^* ? It is easy to see that the combinatorial properties of *composable* blocks do not change with this generalized rewinding scheme. Therefore, we only need to count the number of *composable* blocks in this new rewinding schedule to bound *Sim*'s failure probability. As in Section 2.2.2, we start by counting the number of minimal containing blocks. The following two claims mirror Claim 5 and 6:

Claim 10. *In an execution of *Sim* with splitting factor g on a *bad* random tape, there are at least $k/(g - 1)$ minimal containing blocks.*

Proof. Recall that in an execution with a *bad* tape, there are k convincing slots

of the main session on the main thread (in order to reach `end`). Since each slot has its respective minimal containing block, and any block can be the minimal containing block for at most $g - 1$ slots (by the pigeon hole principle), there are at least $k/(g - 1)$ minimal containing blocks. \square

Claim 11. *Consider an execution of `Sim` with splitting factor g on bad random tape τ . If there are k' minimal containing block, then there are at least $k' - 2 \log_g T$ composable blocks.*

Proof. As shown in Claim 6, it still holds that any minimal containing block that does not contain `start` or `end` is a composable block. Since `start` (or `end`) can only be in at most $\log_g T$ different blocks on the main thread (since that is the recursion depth), we conclude that there are at least $k' - 2 \log_g T$ composable blocks. \square

Following the claims in Section 2.2.2, we conclude that `Sim` with splitting factor g is still a valid zero-knowledge simulator as long as

$$\frac{1}{2^{k/(g-1) - \log_g T}}$$

is negligible in the security parameter n ; this holds whenever $k/g \in \omega(\log n)$. In particular, for any $\varepsilon > 0$, if we set $g = 2^{1/\varepsilon}$ and $k = \omega(\log n)$, then protocol `ConcZKArg` remains secure and $\omega(\log n)$ -rounds, and `Sim` makes at most $T^{1+\varepsilon}$ queries to V^* where T is the maximum number of queries needed in a straight-line interaction with V^* .

2.3 Precise Concurrent Zero-Knowledge

Zero-knowledge is formalized by requiring that the view of any adversarial verifier V^* in an interaction with a prover can be reconstructed by an efficient PPT simulator S , interacting with no one, on input just x . Since whatever V^* “sees” in the interaction can be reconstructed by the simulator, the interaction does not contain anything that V^* cannot already compute by itself (on just the input x). Because the simulator is allowed to be an arbitrary PPT machine, this traditional notion of ZK only guarantees that the *class* of PPT adversaries learn nothing. To measure the knowledge gained by a particular verifier, Goldreich, Micali and Wigderson [28] (see also [24]) put forward the notion of *knowledge tightness*: intuitively, the “tightness” of a simulation is a function relating the (worst-case) running-time of the verifier and the (expected) running-time of the simulator. Thus, in a knowledge-tight ZK proof, the verifier is guaranteed not to gain more knowledge than what it could have computed in time closely related to its *worst-case* running-time.

Micali and Pass [49] recently introduced the notion of *precise zero knowledge* (originally called *local* ZK in [49]). In contrast to traditional ZK (and also knowledge-tight ZK), precise ZK considers the knowledge of an individual verifier in an *individual execution*—it requires that the view of any adversarial verifier V^* , in which V^* takes t computational steps, can be reconstructed in time closely related to t —say $2t$ steps. More generally, we say that a zero-knowledge proof has precision $p(\cdot)$ if the simulator uses at most $p(t) + \text{poly}(n)$ steps to output a view in which V takes t steps on common input an instance $x \in \{0, 1\}^n$.

This notion thus guarantees that the verifier does not learn more than what

can be computed in time closely related to the *actual* time it spent in an interaction with the prover. Such a guarantee is important, for instance, when considering knowledge of “semi-easy” properties of the instance x , considering proofs for “semi-easy” languages L , or when considering *deniability* of interactive protocols (see [49, 52] for more discussion).

In a joint work with Pandey et. al. [51], we construct the first ZK protocol that enjoys both precision and concurrent security. The crux of [51] is a slightly modified KP simulator:

1. The KP simulator is modified to obviously rewind based on time, i.e., the number of Turing machine *steps* taken by V^* , instead of the number of *messages* sent by V^* .
2. The KP simulator is modified to output the view of V^* on the “first” thread that it simulates (i.e., outputs the view in the first sibling block instead of the second).

In fact, the second modification could not be directly analyzed with the techniques of [51] (based on the PRS analysis). Instead, [51] ask that sibling blocks be simulated in parallel (instead of one after another); this requires subtle modifications to the PRS analysis, and the addition of a doubling trick to guess the running time of V^* so that the simulator knows how many recursive levels to simulate in parallel.

In this section we show how precision can be achieved with the new analysis presented in Section 2.2. Looking at the lazy KP simulator, it already outputs the view of V^* on the “first” simulation thread. Therefore, to make the lazy KP simulator precise, we *only* need to modify it to rewind based on time. In

other words, simply let $\text{lazy-rewind}(t, \cdot, \cdot)$ simulate V^* for t Turing machine steps instead of t messages. The observations in Section 2.2.3 then allow us to obtain (and expand to arguments of knowledge) the main theorems of [51], namely:⁷

Theorem 12. *For any integer functions k and g satisfying $k(n)/g(n) \in \omega(\log n)$, there exists a $O(k(n))$ -round concurrent zero-knowledge argument of knowledge for all of NP, based on one-way functions, with precision $p(n, t) \in O(t^{2^{\log_{g(n)} t}})$. In particular, for any $\varepsilon > 0$, there exists a $\omega(\log n)$ -round instantiation of the protocol with precision $p(n, t) \in O(t^{1+\varepsilon})$.*

2.4 The PRS Analysis

In this section we give a brief overview of the PRS analysis, and compare it to our extensions.

On counting arguments. The PRS approach of mapping bad random tapes to good random tapes is different from the approach taken in this paper. In this section, we provide a brief overview of the PRS analysis.

Given a bad random tape, the PRS analysis deals with *minimal rewinding intervals*, defined to be minimal blocks that contain a slot, without containing start or end.⁸ Since minimal rewinding intervals are not “composable” when they overlap, the PRS analysis focuses on a (maximal) set of *disjoint* minimal rewinding intervals. To make up for lost intervals due to overlapping, the PRS analysis

⁷In [49, 51], each verifier message was padded to ensure that the simulator has enough time to generate its messages. We have defined precision to allow the simulator some fixed polynomial overhead (independent of the running time of V^*) to overcome this technical (but simple) obstacle. See [49, 51] for more details.

⁸Here we adopt some of our terminologies to explain the PRS analysis.

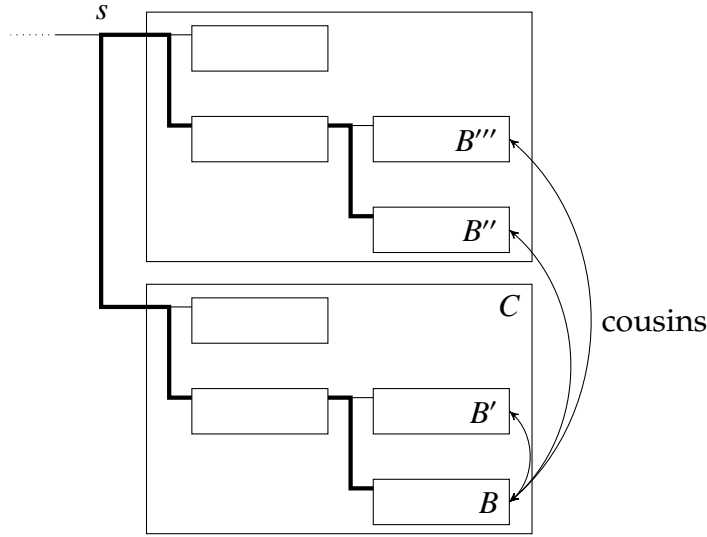


Figure 2.6: A pictorial representation of the original KP rewind schedule, extended from Figure 5 of [61]. We show how a rewinding interval B is related to its sibling (B'), its parent (C), and its cousins (B' , B'' , B'''). To swap block B with its cousin B'' , one needs to exchange the randomness used on the two highlighted thread.

swaps each minimal rewinding interval not only with its sibling (as we do), but also with its “cousins”. See Figure 2.6 for an illustration of cousins blocks. Note that a block may have many cousins (but only one sibling). Moreover, swapping a block with its cousins may require an exchange of random tape segments outside the two blocks, and therefore produce changes in the simulation outside of the cousins

Next, the analysis needs to determine for each rewinding interval, how many cousins swaps will result in a new distinct random tape; this step is complicated because a large portion of the random tape maybe shuffled to perform a cousin swap, destroying other potential rewinding intervals. Each rewinding interval is thus assigned a weight corresponding to the number of available cousins. Finally, a somewhat elaborate analysis is used to lower-bound the sum of weights over the chosen (maximal) subset of disjoint rewinding intervals. (Recall that, in

contrast, our analysis is local—we are only required to show that a *single* swap of a block with its sibling results in *one* new random tape.)

On the “hybrid simulator”. To enable the above counting argument, the PRS analysis only bounds the failure probability of a “hybrid rewinding simulator”, which uses the witness of the input statement to produce a simulated view of V^* . More specifically, the hybrid simulator proceeds like the actual simulator to extract “fake witnesses”—and fails whenever extraction does not work—but uses the real witness to complete the Stage 2 proof. Next, the PRS analysis shows that the hybrid simulator is indistinguishable from the real simulator. This relies on the Stage 2 proof being committed-verifier, so that the adversarial verifier cannot extract the witness used in the Stage 2 proof (and thus distinguish the two simulators), even though there are many rewinds. In contrast, we do not use a hybrid argument regarding two rewinding simulators (see Claim 9), and so our analysis is not limited to committed-verifier protocols.

CHAPTER 3

CONCURRENT ZERO KNOWLEDGE IN THE TIMING MODEL, REVISITED

As a cost for concurrent security, known cZK protocols are usually less efficient than their standalone ZK counterparts. This gap in efficiency, the *concurrency overhead* can take many forms.

The original constant-round cZK protocol of [20] is constructed in the timing model (also explored in [25]). Informally speaking, the timing model assumes that every party has a local stopwatch, and that all these stopwatches more or less agree on how long is 1 second. Also, all parties know a (pessimistic) upper-bound, Δ , on the time it takes to deliver a message on the network. As argued by Goldreich [25], this assumption seems most reasonable for systems today. The problem, however, is that known cZK protocols in the timing model [20, 25] are not very efficient in terms of execution time: despite having a constant number of rounds (4 or 5 messages), the prover in these protocols delays the response of certain messages by time Δ . In other words, every instance of the protocol must take time longer than the pessimistic bound on the max latency of the network (rather than being based on the actual message-delivery time).

Leaving the timing model, Richardson and Kilian [64] (and subsequent improvements by Kilian and Petrank [42] and Prabhakaran, Rosen and Sahai [61]) show how to construct cZK protocols in the standard model (without clocks). Here the protocols are “message-delivery” driven, but there is a significant increase in round-complexity. Whereas constant-round ZK protocols exist in the standalone setting, $\tilde{O}(\log n)$ -rounds are both necessary and sufficient for black-box cZK protocols [61, 43, 65, 14]. To circumvent this lower bound, many constant-

round cZK protocols are constructed under setup assumptions, some more costly than others, such as public-key registries or common reference strings (e.g., [21, 18, 13]). In another related work, Pass and Venkatasubramanian [59] give a constant-round black-box cZK protocol without setup assumptions, but at the expense of having quasi-polynomial time simulators (against quasi-polynomial time adversaries). Finally, known non-black-box ZK protocols are inherently inefficient due to their use of Karp reductions. So far, we have seen the concurrency overhead in the form of extra round complexity, protocol running time, simulator running time (concrete security), or costly setup assumptions.

Overview

In this chapter we revisit the timing model. Ideally, we want to construct cZK protocols that are efficient in all three following areas: small (constant) round-complexity, low imposed delays, and fast simulation (we already have a low cost setup assumption). As communicated by Goldreich [25], Barak and Micciancio suggested the following possible improvement to cZK protocols in the timing model: The prover may only need to impose a delay δ that is a linear fraction of Δ (say $\delta = \Delta/d$), at the expense of increasing the running time of the ZK simulator exponentially (around $n^{O(d)}$). In other words, there could be a compromise between protocol efficiency and knowledge security [24, 49] (i.e., simulator running-time). However, as discussed in [25], this suggestion has not been proven secure. We show that such a trade-off is not only possible, but can be significantly improved.¹

¹It seems that traditional techniques can be used to demonstrate the Barak-Micciancio suggestion when the adversary employs a *static* scheduling of messages. However, *adaptive* schedules seem harder to deal with. See Section 3.2.1 for more details.

Trading rounds for minimum delays. The original work of Richardson and Kilian [64] shows that increasing the number of communication rounds can decrease the running-time of the simulator. Our first result shows that by only slightly increasing the number of rounds, but still keeping it constant (e.g., 10 messages), the prover may reduce the imposed delay to $\delta = \Delta/2^d$, while keeping the simulator running time at $n^{O(d)}$. This is accomplished by combining simulation techniques from both the timing model [20, 25] (polynomial time simulation but high timing constraints) and the standard model [64, 59] (quasi-polynomial time simulation but no timing constraints). As far as we know, this yields the first formal proof that constant-round concurrent zero-knowledge protocols are possible using a delay δ that is smaller than Δ .

“Eye-for-an-eye” delays. The traditional approach for constructing cZK protocols is to “penalize” all parties equally, whether it is in the form of added round complexity or imposed timing delays. One may instead consider the notion of punishing only adversarial behavior, similar to the well-known “tit-for-tat” or “eye-for-an-eye” technique of game theory (see e.g., [1]). The work of Cohen, Kilian and Petrank [16] first implemented such a strategy (with respect to cZK) using an iterated protocol where in each iteration, the verifier is given a time constraint under which it must produce all of its messages; should a verifier exceed this constraint, the protocol is restarted with doubled the allowed time constraint (the punishment here is the resetting); their protocol had $\tilde{O}(\log^2 n)$ rounds and $\tilde{O}(\log n)$ “responsive complexity”—namely, the protocol takes time $\tilde{O}(\log n)T$ to complete if each verifier message is sent within time T . The work of Persiano and Visconti [60] and Rosen and shelat [67] takes a different approach and punish adversaries that perform “bad” schedulings of messages by

adaptively adding more rounds to the protocol; their approaches, however, only work under the assumption that there is a *single* prover, or alternatively that all messages on the network are exposed on a broadcast channel (so that the provers can check if a problematic scheduling of messages has occurred).

In our work, we instead suggest the following simple approach: Should a verifier provide its messages with delay t , the prover will delay its message accordingly so that the protocol completes in time $p(t) + \delta$, where p is some *penalty* function and δ is some small minimal delay. We note that, at a high-level, this approach is somewhat reminiscent of how message delivery is performed in TCP/IP.

As we show, such penalty-based *adaptive delays* may significantly improve the compromise between protocol efficiency and knowledge security. For example, setting $p(t) = 2t$ (i.e., against a verifier that responds in time $t < \Delta$, the prover responds in time $t + \delta$) has a similar effect as increasing the number of rounds: The prover may reduce the minimal imposed delay to $\delta = \Delta/2^d$, while keeping the simulator running time at $n^{O(d)}$. Moreover, if we are willing to use more aggressive penalty functions, such as $p(t) = t^2$, the minimal delay may be drastically reduced to $\delta = \Delta^{1/2^d}$, greatly benefiting “honest” parties that respond quickly, while keeping the same simulator running time. Note that, perhaps surprisingly, we show that such a “tit-for-tat” technique, which is usually employed in the setting of rational players, provides significant efficiency improvements even with respect to fully adversarial players.

Combining it all. Finally, we combine our techniques by both slightly increasing the round complexity and implementing penalty-based delays. We state our

main theorem below for $p(t) = t$ (no penalty), ct (linear penalty), and t^c (polynomial penalty) (in the main text we provide an expression for a generic $p(t)$):

Theorem 13. *Let Δ be an upper-bound on the time it takes to deliver a message on the network. Let r and d be integer parameters, and $p(t)$ be a (penalty) function. Then, assuming the existence of claw-free permutations, there is a $(2r + 6)$ -message black-box perfect cZK argument for all of \mathbf{NP} with the following properties:*

- *The simulator has running time $(rn)^{O(d)}$.*
- *For any verifier that cumulatively delays its message by time at most T , the prover will provide its last message in time at most $p(T) + \delta$, where*

$$\delta = \begin{cases} 2\Delta/r^d & \text{if } p(t) = t \quad (\text{no penalty}) \\ 2\Delta/(cr)^d & \text{if } p(t) = ct \quad (\text{linear penalty}) \\ \frac{(2\Delta)^{1/c^d}}{r^{1+1/c+\dots+1/c^{d-1}}} \leq \frac{(2\Delta)^{1/c^d}}{r} & \text{if } p(t) = t^c \quad (\text{polynomial penalty}) \end{cases}$$

We make a few observations regarding Theorem 13.

On the number of rounds. Even without penalty-based delays, if $r = 2$, we achieve an exponential improvement in the imposed delay ($\delta = \Delta/2^d$), compared to the Barak-Micciancio suggestion (as communicated by Goldreich [25]) which suggested a delay of $\delta = \Delta/d$. Larger r (i.e., more rounds) allows us to further improve the delay.

On adversarial networks. If an adversary controls the whole network, it may also delay messages from the honest players. In this case, honest players (that answer as fast as they can) are also penalized. However, the adversary can anyway delay message delivery to honest players, so this problem is unavoidable. What we guarantee is that, if a pair of honest players are communication over a

channel that is not delayed (or only slightly delayed) by the adversary, then the protocol will complete fast.

On networks with failure. Note that even if the network is not under adversarial control, messages from honest parties might be delayed due to network failures. We leave it as an open question to (experimentally or otherwise) determine the “right” amount of penalty to employ in real-life networks: Aggressive delays allow us to minimize the imposed delay δ , but can raise the expected protocol running time if network failures are common. Note that one solution is for honest parties to restart their protocol sessions in the presence of network spikes; such restarts would harm an adversary staging a coordinated attack, and are relatively harmless towards honest parties (so long as network spikes are rare).

On concurrent multi-party computation. [40] and [45] show that concurrent multi-party computation (MPC) is possible in the timing model using delays of length $O(\Delta)$. Additionally, [45] shows that at least $\Delta/2$ delays are necessary to achieve concurrent MPC in the timing model. In retrospect, this separation between concurrent ZK and MPC should not be surprising since cZK can be constructed in the plain model [64, 42, 61], but concurrent MPC cannot [12, 47].

Organization

In Section 3.1 we give definitions regarding the timing model. An overview of our protocol and zero-knowledge simulator, followed by their formal descriptions, is given in Section 3.2. The analysis of our protocol and simulator is found

in Section 3.3.

3.1 The Timing Model

In the timing model, originally introduced by Dwork, Naor and Sahai [20], we consider a model that incorporates a “timed” network. Informally, in such a network, a (known) maximum network latency Δ —the time it takes for a message to be computed and delivered over the network—is assumed. Moreover, each party (in our case the honest provers) possesses a local clock that is somewhat synchronized with the others (in the sense that a second takes about the same time on each clock).

As in [20, 25, 40], we model all the parties in the timing model as interactive Turing machines that have an extra input tape, called the **clock tape**. In an adversarial model, the adversary has full control of the content of everyone’s clock tape (it can initialize and update the tape value at will), while each machine only has read access to its own clock tape. More precisely, when a party P_i is invoked, the adversary initializes the local clock of P_i to some time t of its choice. Thereafter the adversary may, at any time, overwrite the all existing clock tapes with new time values. To model that in reality most clocks are reasonably but not perfectly synchronized, we consider adversaries that are ε -**drift preserving**, as defined below:

Let $\sigma_1, \sigma_2, \dots$ be a series of global states of all machines in play; these states are recorded whenever the adversary initiates a new clock or updates the existing clocks. Denote by $\text{CLK}_P(\sigma)$ the value of the local clock tape of machine P at state σ . We say that an adversary is ε -**drift preserving** if for every pair of parties

P and P' and every pair of states σ and σ' , it holds that

$$\frac{1}{\varepsilon}(\text{CLK}_P(\sigma) - \text{CLK}_P(\sigma')) \leq \text{CLK}_{P'}(\sigma) - \text{CLK}_{P'}(\sigma') \leq \varepsilon(\text{CLK}_P(\sigma) - \text{CLK}_P(\sigma'))$$

As in [20, 25, 40], we use the following constructs that utilize the clock tapes. Below, by local time we mean the value of the local clock tape.

Delays: When a party is instructed to **delay** sending a message m by δ time, it records the present local time t , checks its local clock every time it is updated, and sends the message when the local time reaches $t + \delta$.

Time-out: When a party is instructed to **time-out** if a response from some other party P_i does not arrive in δ time, it records the present time t . When the message from P_i does arrive, it aborts if the local time is greater than $t + \delta$.

Measure: When a party is instructed to **measure** the time elapsed between two messages, it simply reads the local time t when the first message is sent/received, and reads the local time t' again when the second message is sent/received. The party then outputs the elapsed time $t' - t$.

Although the **measure** operator is not present in previous works, it is essentially the quantitative version of the **time-out** operation, and can be implemented without extending the timing model. For simplicity, we focus on the model where the adversary is 1-drift preserving, i.e. all clocks are synchronized, but our results easily extend to ε -drift preserving adversaries.

3.1.1 Black-Box Concurrent Zero-Knowledge in the Timing Model

The standard notion of concurrent zero-knowledge extends straightforwardly to the timing model; all machines involved are simply augmented with the aforementioned clock tape. The view of a party in an interaction still consists of all incoming messages as well as the its random tape, and additionally contains its clock tape. In particular, the view of the adversary determines the value of all the clocks.

Our definition of cZK in the timing model is slightly different from that of [25]. The definition of [25] assumes that the adversary never triggers a time-out from any prover. [25] also assumes that the adversary always delays the verifier messages as much as permitted, but this assumption is no longer without loss of generality for protocols with penalty-based delays. Therefore in our model, the adversary is given total control over all the clocks (subject to ε -drift preserving), similar to the definition of [40] for the setting of concurrent multi-party computation.

3.2 A cZK Protocol a Simulator in the Timing Model

Following the works of [23, 26], later extended to the concurrent setting by [64, 42, 61, 59], we consider ZK protocols with two stages:

Stage 1: First the verifier V “commits to a trapdoor” (the start message). This is followed by one or multiple **slots**; each slot consists of a prover challenge (the opening of the slot) followed by a verifier response (the closing of the slot). A rewinding black-box ZK simulator can rewind any one of these

slots to extract the verifier trapdoor.

Stage 2: The protocol ends with a modified proof of the original statement that can be simulated given the verifier trapdoor.

To generate the view of an adversarial verifier V^* in the standalone setting, a black-box simulator simply rewinds a slot to learn the trapdoor, and use it to simulate the final modified proof.

In the concurrent setting, however, V^* may *fully nest* another session inside a slot (i.e., after the prover sends the opening message, V^* schedules a full session before replying with the closing message). In order for the simulator to rewind this slot, it would need to simulate the view of the nested session twice. Therefore, repeated nesting may cause a naive simulator to have super-polynomial running time [20]. Different techniques were employed in different models to circumvent this difficulty caused by nesting. In the timing model, [20, 25] shows that by delaying the Stage 2 proof and limiting the time allowed between the opening and closing of any slot, we can avoid the nesting situation all together. On the other hand, [64] showed that if the protocol has enough slots, the simulator can always find a slot that isn't "too nested" to rewind.

3.2.1 The Protocol Intuition

The work of [59] describes a simulator (based on the work of [64]) that works also for constant-round protocols. Its running time (implicitly) depends on the maximum nesting level/depth of the least nested slot. Specifically, the running time of the simulator is $n^{O(d)}$ when this maximum depth of nesting is d . Building

upon this, we now focus on reducing the maximum depth of nesting in the timing model.

In the following overview of our techniques, we assume that V^* interleaves different sessions in a *static* schedule; the full generality of dynamic scheduling is left for our formal analysis. Additionally, we keep track of the running time of our protocol as a function of T —the total amount of accumulated delay caused by the verifier in all the messages.

Imposing traditional timing delays with one slot. We first review the works of [20, 25]. Recall that Δ is the maximum network latency—the time it takes for a message to be computed and delivered over the network. We require that the time between the opening and closing of each slot be bounded by 2Δ (otherwise the prover aborts); this is the smallest time-out value that we may ask of the honest verifier. At the same time, the prover delays the Stage 2 proof by δ time (after receiving the closing message of the last slot), where δ is a parameter (Fig. 3.1a). It is easy to see that if $\delta = 2\Delta$, then no nesting can occur (Fig. 3.1b). In this case the running time of the protocol is $T + \Delta$.

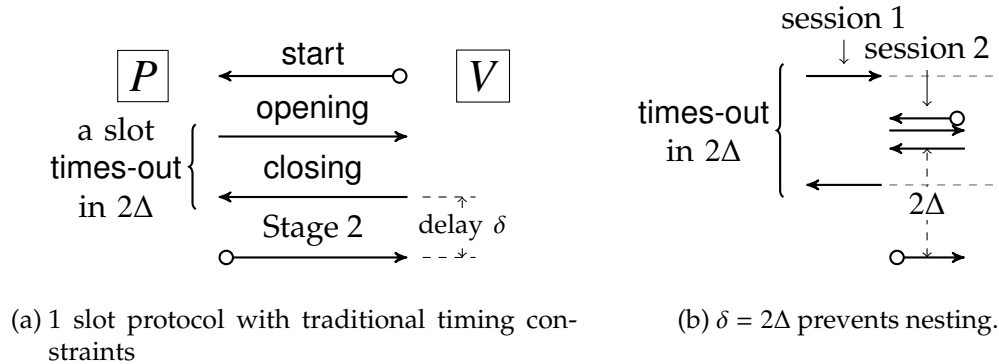


Figure 3.1: Traditional timing delays with **1 slot**.

If we consider the suggestion of Barak and Micciancio and set $\delta = 2\Delta/d$, then

up to d levels of nesting can occur (Fig. 3.2). In this case, the running time of the protocol is $T + 2\Delta$ and $T + 2\Delta/d$, respectively.

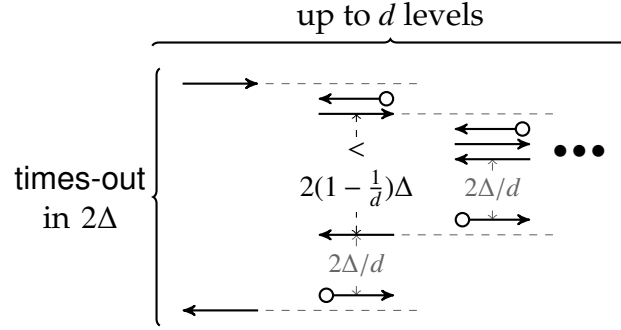


Figure 3.2: $\delta = 2\Delta/d$ gives at most d levels of nesting.

Increasing the number of slots. This idea was first explored by [64] in the standard model where intuitively, more slots translates to more rewinding opportunities for the simulator. In the timing model, the effect of multiple slots is much more direct. Let us look at the case of 2 slots. Suppose in some session, V^* delays the closing of a slot by the maximum allowed time, 2Δ . Further suppose that V^* nests an entire session inside this slot. Then in this nested session, one of the slots must have taken time less than Δ (Figure 3.3a). Continuing this argument, some fully nested session at level d must take time less than $2\Delta/2^d$. Therefore if we set $\delta = 2\Delta/2^d$, V^* cannot fully nest every slot beyond depth d , and the running time of the protocol becomes $T + 2\Delta/2^d$.

Penalizing the adversarial verifier with adaptive delays. Here we implement our “eye-for-an-eye” approach of penalizing adversarial verifiers that delay messages. Let $p(t)$ be a **penalty function** that satisfies $p(t) > t$ and is monotonically increasing. During Stage 1 of the protocol, the prover measures t , the total time elapsed from the opening of the first slot to the closing of the last slot. Based on

this measurement, the prover **delays** Stage 2 by time $p(t) - t$ or by the minimal imposed delay δ , whichever is greater. As a result, Stage 2 only starts after $p(t)$ time has elapsed starting from the opening of the first slot. For example, suppose $p(t) = 2t$ and that the protocol has 1 slot. Then for V^* to fully nest a session inside a slot that took time 2Δ , the slot of the nested session must have taken time at most Δ , giving the same effect as having 2 slots (Fig. 3.3b). Furthermore, if we implement more aggressive penalties, such as $p(t) = t^2$,² then the slot of the nested session is reduced to time $\sqrt{2\Delta}$. Therefore if we set $\delta = (2\Delta)^{1/2^d}$, V^* cannot fully nest every slot beyond depth d , and the running time of the protocol becomes $T^2 + (2\Delta)^{1/2^d}$.

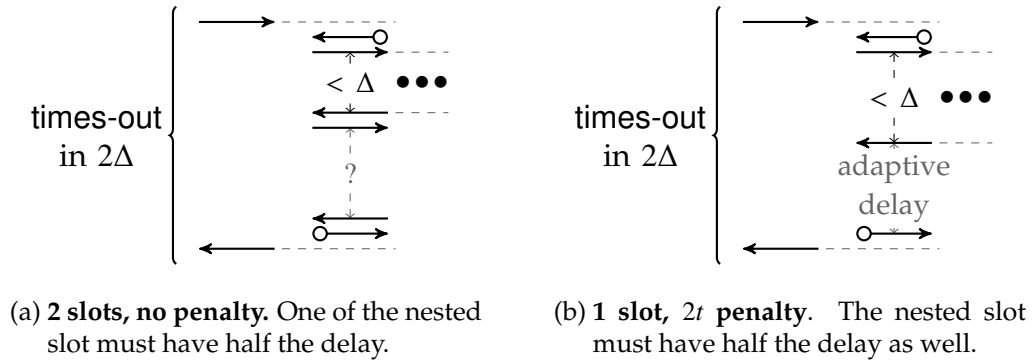


Figure 3.3: Our main techniques of restricting the nesting depth of V^* .

Combining the techniques. In general, we can consider concurrent ZK protocols that both contain multiple slots and impose penalty-based delays (e.g., Fig. 3.4). If we have r slots and impose $p(t)$ penalty on delays, and define

²Formally we may use $p(t) = t^2 + 1$ to ensure that $p(t) > t$.

$g(t) = p(rt)$, then δ can be decreased to

$$d \text{ times } \left\{ \frac{p^{-1}\left(\dots \frac{p^{-1}\left(\frac{p^{-1}(2\Delta)}{r}\right)}{r}\right)}{r} = (g^{-1})^d(2\Delta) \right.$$

$$= \begin{cases} 2\Delta/r^d & \text{if } p(t) = t \quad (\text{no penalty}) \\ 2\Delta/(cr)^d & \text{if } p(t) = ct \quad (\text{linear penalty}) \\ \frac{(2\Delta)^{1/c^d}}{r^{1+1/c+\dots+1/c^{d-1}}} \leq \frac{(2\Delta)^{1/c^d}}{r} & \text{if } p(t) = t^c \quad (\text{polynomial penalty}) \end{cases}$$

while keeping the simulator running time at $(rn)^{O(d)}$. The running time of the protocol is then $p(T) + \delta$.

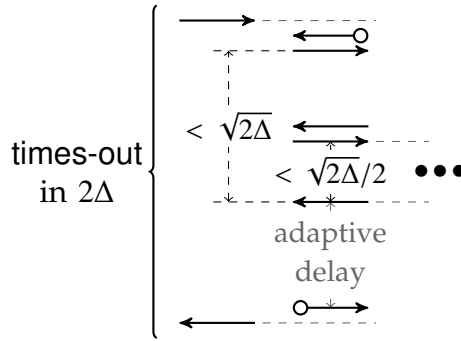


Figure 3.4: **2 slots and t^2 penalty.** Slots of nesting sessions decrease in size very quickly.

Handling dynamic scheduling. So far we have discussed our analysis (and have drawn our diagrams) assuming that V^* follows a static schedule when interleaving multiple sessions. In general though, V^* may change the scheduling dynamically based on the content of the prover messages. As a result, the schedule (and nesting) of messages may change drastically when a black-box simulator rewinds V^* . This phenomenon introduces many technical difficulties into the analysis, but fortunately the same difficulties were also present and resolved [59]. By adapting the analysis in [59], we give essentially the same results

in the case of dynamic scheduling, with one modification: An additional slot is needed whenever $\delta < 2\Delta$ (this includes even the case illustrated in Fig. 3.2). For example, a minimal of 2 slots is needed to implement penalty-based delays, and a minimum of 3 slots is needed to reap the improvements that result from multiple slots.

Handling ε -drifts in clock tapes. As in the work of [20, 25] we merely need to scale the time-out values in our protocols when the local clocks are not perfectly synchronized. Specifically, if the adversary is ε -drift preserving for some $\varepsilon \geq 1$, then our protocol will impose a minimal delay of $\varepsilon\delta$ and an adaptive delay of $\varepsilon p(t)$ (when applicable) between the closing of the last slot and Stage 2.

3.2.2 Description of the protocol

Our cZK argument in the timing model, protocol TIMINGCONCZKARG, is very similar to our cZK argument in the plain model, protocol CONCZKARG (see Figure 2.1).

Let L be a language in NP with witness relation \mathbf{R}_L . Given a one-way function f , a parameter r , a penalty function $p(t) : \mathbb{N} \rightarrow \mathbb{N}$ that is monotonically increasing and satisfies $p(t) > t$, a maximum delay Δ and a minimal delay δ , TIMINGCONCZKARG for language L proceeds in the following two stages. On common input a security parameter n and an instance $x \in \{0, 1\}^n$:

Stage 1: The verifier picks two random strings $s_1, s_2 \in \{0, 1\}^n$ and sends $c_1 = f(s_1), c_2 = f(s_2)$ to the prover. The verifier also sends $\alpha_1, \dots, \alpha_{r+1}$, the first

messages of $r + 1$ invocations of a WI special-sound proof of the statement “ c_1 and c_2 are in the image set of f ”. These proofs are then completed sequentially in $r + 1$ iterations.

In the j^{th} iteration, the prover first sends $\beta_j \leftarrow \{0, 1\}^{n^2}$, a random second message for the j^{th} proof (opening of the j^{th} slot), then the verifier replies with the third message γ_j of the j^{th} proof (closing of the j^{th} slot). The prover times-out the closing of each slot with time 2Δ , and measures the time that elapsed between the opening of the first slot and the closing of the $r + 1^{\text{st}}$ slot as t .

Stage 2: The prover delays by time $\max\{p(t) - t, \delta\}$, and then provides a perfect-WI proof of knowledge of the statement “either $x \in L$, or that (at least) one of c_1 and c_2 are in the image set of f ”.

Protocol TIMINGCONCZKARG is more formally described in Figure 3.5. Observe that in addition to the timing operations, there are two small differences between protocol TIMINGCONCZKARG and CONCZKARG,

- The opening of each slot (the third message of the SS proof) has length n in CONCZKARG, and length n^2 in TIMINGCONCZKARG. As we will see, the n^2 length requirement enables us to achieve perfect zero-knowledge. (Also note that this requirement can be achieved with suitable padding.)
- Protocol CONCZKARG contains k slots, while protocol TIMINGCONCZKARG contains $r + 1$ slots. This change of “round complexity parameter” is more convenient for the analysis TIMINGCONCZKARG.
- Protocol TIMINGCONCZKARG uses a perfect WI argument in Stage 2. This allows the simulator to generate the view of an adversarial verifier ex-

Protocol TIMINGCONCZKARG:

Common Input: an instance x of a language L with witness relation \mathbf{R}_L .

Auxiliary Input for Prover: a witness w , such that $(x, w) \in \mathbf{R}_L(x)$.

Parameters: r (round complexity), p (penalty function), Δ (max delay), δ (min delay)

Stage 1:

V uniformly chooses $s_1, s_2 \in \{0, 1\}^n$.

V \rightarrow P: $c_1 = f(s_1)$, $c_2 = f(s_2)$, and $r + 1$ first messages $\alpha_1, \dots, \alpha_{r+1}$ of 4-round computational-WI and SS proofs on common input (c_1, c_2) with respect to the witness relation:

$$\mathbf{R}_f(c_1, c_2) = \{s : f(s) = c_1 \text{ or } f(s) = c_2\}$$

This is called the **start message**.

For $j = 1$ to $r + 1$ do

P \rightarrow V [opening of slot j]: Select a second message $\beta_j \leftarrow \{0, 1\}^{n^2}$ for the j^{th} WI special-sound proof. P times-out if the next verifier message is not received in time 2Δ .

V \rightarrow P [closing of slot j]: Third message γ_j for the j^{th} WI special-sound proof.

P measures the time elapsed between the opening of the first slot and the closing of the $r + 1^{\text{st}}$ slot as t .

Stage 2:

P delays the next message by time $\max\{p(t) - t, \delta\}$.

P \leftrightarrow V: A perfect-WI argument of knowledge on input (x, c_1, c_2) with respect to the witness relation:

$$\mathbf{R}_{f \vee L}(c_1, c_2, x) = \{(s, w) : s \in \mathbf{R}_f(c_1, c_2) \text{ or } w \in \mathbf{R}_L(x)\}$$

Figure 3.5: A concurrent ZK argument for NP in the timing model.

actly (sometimes called perfect zero-knowledge), which in turn enables our analysis.

The soundness and the completeness of the protocol follows directly from the proof of Feige and Shamir [23]; in fact, the protocol is an instantiation of theirs. Intuitively, to cheat in the protocol a prover must “know” an inverse to

either c_1 or c_2 , which requires inverting the one-way function f .

3.2.3 Simulator Overview

At a very high-level our simulator follows that of Feige and Shamir [23]. The simulator will attempt to rewind one of the special-sound proofs (i.e., the slots), because whenever the simulator obtains two accepting proof transcripts, the special-soundness property allows the simulator to extract a “fake witness” r_i such that $c_i = f(r_i)$. This witness can later be used in the second phase of the protocol. At any point in the simulation, we call a session of the protocol **solved** if such a witness has been extracted. On the other hand, if the simulation reaches Stage 2 of a session without extracting any “fake witnesses”, we say the simulation is **stuck**.

In more detail, our simulator is essentially identical to that of [59], which in turn is based on the simulator of [64]. The general strategy of the simulator is to find and rewind the “easiest” slot for each session; during a rewind, the simulator recursively invokes itself on any nested sessions when necessary. The main difference between our work and that of [64, 59] lies in determining which slot to rewind. In [64, 59], a slot that contains a “small” amount of **start** messages (freshly started sessions) is chosen, whereas in our simulation, a slot that runs in “*little time*” (between the **opening** and the **closing**) is rewound. As we will see, part of the analysis from [59] applies directly to our simulator modulo some changes in parameters; we only need to ensure that our definition of “little time” allows the simulator to always rewind some slot (formally argued in Claim 14).

3.2.4 Description of the simulator

Our simulator is defined recursively. Intuitively on recursive level 0, the simulator's goal is to generate a view of V^* , while on all other recursive levels, the simulator's goal is to rewind a particular slot (from a previous recursion level). On recursive level ℓ , the simulator starts by feeding random Stage 1 messages to V^* . Whenever a slot s closes, S decides whether or not to rewind s depending on the time elapsed between the opening and the closing of s . If the elapsed time is "small" (where the definition of small depends on the level ℓ), S begins to rewind the slot. That is, S recursively invokes itself on level $\ell + 1$ starting from the opening of slot s with a new (random) message β , with the goal of reaching the closing message of slot s . While in level $\ell + 1$, S continues the simulation until one of the following happens:

1. *The closing message γ for slot s occurs:* S extracts a "fake" witness using the special-sound property and continues its simulation (on level ℓ).
2. *V^* aborts or delays "too much" in the rewinding:* S restarts its rewinding using a new challenge β for s . We show in expectation, S only restarts $O(1)$ times. Intuitively, this follows since during the execution at level ℓ , S would only rewind the slot s if V^* did not abort and only took "little time" ("if it happened once, it will happen again").
3. *S is "stuck" at Stage 2 of an unsolved session that started at level $\ell + 1$:* S halts and outputs fail (we later show that this never happens).
4. *S is "stuck" at Stage 2 of an unsolved session that started at level ℓ :* Again, S restarts its rewinding. We show that this case can happen at most $m - 1$ times, where m is the total number of sessions.

5. S is “stuck” at Stage 2 of an unsolved session that started at level $\ell' < \ell$: S returns the view to level ℓ' (intuitively, this is just case 4 for the recursion at level ℓ').

In the unlikely event that S asks the same challenge β twice, S performs a brute-force search for the witness. Furthermore, to simplify the analysis of the running-time, the simulation is cut-off if it runs “too long” and S extracts witnesses for each session using brute-force search.

The basic idea behind the simulation is similar to [59]: We wish to define “little time” appropriately, so that some slot of every session is rewound and that expected running time is bounded. For a technical reason (used later in Claim 14), we actually want the simulator to rewind one of the first r (out of $r + 1$) slots of each session.

Take for example $p(t) = 2t$ and $r = 2$ (3 slots). Based on our intuition from Sect. 3.2.1, a good approach would be to ensure that the simulation at recursive level ℓ finishes within time $2\Delta/4^\ell$, and define “little time” on level ℓ to be $2\Delta/4^{\ell+1}$. Then, we know that any session that is fully executed at recursive level ℓ must have taken time less than $2\Delta/(4^\ell \cdot 2)$ in Stage 1 (due to penalty-based delays), and therefore one of the first two slot must have taken time less than $2\Delta/4^{\ell+1}$, making it eligible for rewind. To show that the expected running time is bounded, we simply set δ appropriately (as a function of d , Δ and r) as in Sect. 3.2.1, and this would guarantee that the recursion depth of the simulator is bounded.

A formal description of our simulator can be found in Figure 3.6. We rely on the following notation.

- Define the function $g : \mathbb{N} \rightarrow \mathbb{N}$ by $g(n) = p(rn)$. Recall that $g^k(n)$ is the

function computed by composing g together k times, i.e., $g^k(n) = g(g^{k-1}(n))$ and $g^0(n) = n$. Let d (the maximum depth of recursion) be $\min_d \{g^d(\delta) > 2\Delta\}$. Note that if we choose $\delta = (g^{-1})^k(2\Delta)$ for some k , then $d = k$.

- $\text{slot}(i, j)$ will denote slot j of session i .
- W is a repository that stores the witness for each session. The `update W` command extracts a witness from two transcripts of a slot (using the special-sound property). If the two transcripts are identical (i.e. the openings of the slot are the same), the simulator performs a brute-force search to extract a “fake” witness s_i s.t. $c_i = f(s_i)$ for $i \in \{1, 2\}$.
- R is a repository that stores the transcripts of slots of unsolved sessions. Transcripts are stored in R when the simulator gets stuck in a rewinding (cases 4 and 5 mentioned in the high-level description).

3.3 Analysis of the Simulator

To prove correctness of the simulator, we show that the output of the simulator is correctly distributed and its expected running-time is bounded. We first prove in Claim 14 that the simulator never outputs fail. Using Claim 14, we show that the output distribution of the simulator is correct in Claim 15, and that the expected running time of the simulator is at most $\text{poly}(m^d r^d)$ in Claim 18. Theorem 13 then follows from Claim 15 and 18, together with the fact that if $\delta = (g^{-1})^k(2\Delta)$ then $d = k$.

Claim 14 is given below, while Claim 15 and 18 are given in the next two sections; in any case, the proofs of Claim 15 and 18 are essentially identical

SOLVE_d^{V*}($x, \ell, h_{\text{initial}}, s, W, R$):

Let $h \leftarrow h_{\text{initial}}$. Note that h_{initial} contains all sessions that are started on previous recursion levels.

Repeat forever:

1. If v is a Stage 2 verifier message of some session, continue.
2. If V^* aborts in the sessions of slot s , or the time elapsed since h_{initial} exceeds $g^{d+1-\ell}(\delta)$, restart SOLVE from h_{initial} .
3. If the next scheduled message is a Stage 2 prover message for session i and $W(i) \neq \perp$, then use $W(i)$ to complete the WI proof of knowledge; if $W(i) = \perp$ and start message of session i is in h_{initial} return h , otherwise halt with output **fail**.
4. If the next scheduled message is a Stage 1 prover message for slot s' , pick a random message $\beta \leftarrow \{0, 1\}^{n^2}$. Append β to h . Let $v \leftarrow V^*(h)$.
5. Otherwise, if v is the closing message for $s' = \text{slot}(i', j')$, then update W with v (using R) and proceed as follows.
 - (a) If $s = s'$, then return h .
 - (b) Otherwise, if session i' starts in h_{initial} , then return h .
 - (c) Otherwise, if $W(i') \neq \perp$ or the time elapsed since the opening of slot (i', j') exceeds $g^{d-\ell}$, then continue.
 - (d) Otherwise, let h' be the prefix of the history h where the prover message for s' is generated. Set $R' \leftarrow \phi$. Repeat the following m times:
 - i. $h^* \leftarrow \text{SOLVE}_d^{V^*}(x, \ell + 1, h', s', W, R')$
 - ii. If h^* contains an accepting proof transcript for slot s' , extract witness for session i' from h and h^* and update W .
 - iii. Otherwise, if the last message in h^* is the closing message for the last slot of an session that started in h_{initial} return h^* .
 - iv. Otherwise, add h^* to R' .

$S^{V^*(x,z)}(x)$:

Let $d \leftarrow \min_d \{g^d(\delta) > 2\Delta\}$. Run $\text{SOLVE}_d^{V^*}(x, 0, \dots)$ and output whatever SOLVE outputs with one exception. If an execution of $\text{SOLVE}_d^{V^*}(x, 0, \dots)$ queries V^* more that 2^n times, proceed as follows:

Let h denote the view reached in the “main-line” simulation (i.e., in the top-level of the recursion). Continue the simulation in a “straight-line” fashion from h by using a brute-force search to find a “fake” witness each time Stage 2 of an session i is reached.

Figure 3.6: Description of our black-box ZK simulator.

to [59], modulo a change of parameters. Throughout the analysis we assume without loss of generality that the adversary verifier V^* is deterministic (as it can always get its random coins as part of the auxiliary input).

Claim 14. *For every $x \in L$, $S^{V^*(x,z)}(x)$ never outputs fail.*

Proof. Recall that $S^{V^*(x,z)}(x)$ outputs fail only if $\text{SOLVE}_d^{V^*}(x, 0, \dots)$ outputs fail. Furthermore, SOLVE outputs fail at recursive level ℓ only if it reaches Stage 2 of an unsolved session that started at level ℓ (see Step 3 of SOLVE). We complete the proof in two parts. First we show $\text{SOLVE}_d^{V^*}$ will rewind at least one of the first r slots of every session at level ℓ . Then, we show that SOLVE always extracts a witness when it rewinds a slot.

In order for SOLVE to be stuck at a session i that starts at recursive level ℓ , session i must reach Stage 2 within $g^{(d-\ell)}(\delta)$ time-steps (otherwise SOLVE would have rewound as per Step 2). This implies that t , the time between the opening of the first slot and the closing of the last slot of session i , must satisfy $p(t) \leq g^{(d-\ell)}(\delta)$ (due to penalty-based delays). This in turn implies that one of the first r slots of session i must have taking time at most

$$\frac{t}{r} \leq \frac{p^{-1}(g^{(d-\ell)}(\delta))}{r} \leq g^{(d-\ell-1)}(\delta)$$

(here we use the monotonicity of p). By construction, SOLVE would have rewound this slot (i.e., execute Step 5.(d)).

Next we show that whenever SOLVE rewinds a slot, a witness for that session is extracted. Assume for contradiction that SOLVE fails to extract a witness after rewinding a particular slot. Let level ℓ and slot j of session i be the first time this happens. This means at the end of Step 5.(d), m views are obtained, yet none of them contained a second transcript for slot j . Observe that in such

a view, SOLVE must have encountered Stage 2 of some unsolved session i' (i.e., stuck). Yet, we can show that the $m - 1$ other sessions can each cause SOLVE to be stuck at most once; this contradicts the fact that SOLVE is stuck on all m good views.

For every session i' that SOLVE gets stuck on, both the opening and the closing of the last slot occurs inside the rewinding of $\text{slot}(i, j)$; otherwise, SOLVE would have rewound one of the r slots that occurred before the opening of $\text{slot}(i, j)$ successfully and extracted a witness for session i' (i, j was the first “failed” slot). Furthermore, the transcript of this slot enables SOLVE to never get stuck on session i' again, since the next time that the last slot of session i' closes will allow SOLVE to extract a witness for session i' . \square

3.3.1 Indistinguishability of the Simulation

Claim 15. *The following ensembles are identical:*

$$\{\text{View}_{V^*}(P(x, w), V^*(x, z))\}_{x \in L, w \in \mathbf{R}_L(x), z \in \{0, 1\}^*} = \{S^{V^*(x, z)}(x)\}_{x \in L, w \in \mathbf{R}_L(x), z \in \{0, 1\}^*}$$

Proof. Consider the following hybrid simulator \tilde{S} that receives the real witness w to the statement x . \tilde{S} on input x and w proceeds just like S in order to generate the prover messages in Stage 1, but proceeds as the honest prover using the witness w in order to generate messages in Stage 2 (instead of using the “fake” witness as S would have). Using the same proof as in Claim 14, we can show that \tilde{S} never outputs fail. Furthermore, as the prover messages in Stage 1 are chosen uniformly and \tilde{S} behaves like an honest prover in Stage 2. Therefore, we get:

Claim 16. *The following ensembles are identical:*

$$\{\text{View}_{V^*}(P(x, w), V^*(x, z))\}_{x \in L, w \in \mathbf{R}_L(x), z \in \{0,1\}^*} = \{\tilde{S}^{V^*(x,z)}(x, w)\}_{x \in L, w \in \mathbf{R}_L(x), z \in \{0,1\}^*}$$

It remains to show that the output distributions of \tilde{S} and S are identical. This follows from the perfect-WI property of Stage 2 of the protocols, since the only difference between the simulators \tilde{S} and S is the choice of witness used. For completeness, we provide a proof below.

Claim 17. *The following ensembles are identical:*

$$\{S^{V^*(x,z)}(x, w)\}_{x \in L, w \in \mathbf{R}_L(x), z \in \{0,1\}^*} = \{\tilde{S}^{V^*(x,z)}(x, w)\}_{x \in L, w \in \mathbf{R}_L(x), z \in \{0,1\}^*}$$

Proof. To prove the claim we will rely on the fact that the running time of the simulator is bounded. This holds since S stops executing SOLVE whenever it performs more than 2^n queries, and instead continues the simulation a straight-line fashion, extracting “fake” witnesses using brute-force search. Assume for contradiction that the claim is false, i.e., there exists a deterministic verifier V^* such that the ensembles are not identical.

We consider several hybrid simulators, S_i for $i = 0$ to N , where N is an upper-bound on the running time of the simulator. S_i receives the real witness w to the statement x and behaves exactly like S , with the exception that Stage 2 messages in the first i sessions are generated using the honest prover strategy (and the witness w). By construction, $S_0 = \tilde{S}$ and $S_N = S$. By assumption the outputs of S_1 and S_N are not identically distributed, therefore there must exist some j such that the output of S_j and S_{j+1} are different. But S_j and S_{j+1} differs only in the witness used to provide the Stage 2 argument of the $j + 1^{\text{st}}$ session; this contradicts the perfect-WI property of the Stage 2 argument. \square

□

3.3.2 Running Time of the Simulator

Claim 18. *For all $x \in L, z \in \{0, 1\}^*$, and all m -session concurrent adversarial verifier V^* , the expected running time of $S^{V^*(x,z)}(x)$ is bounded by $\text{poly}(m^d r^d)$.*

Proof. Recall that S starts by running SOLVE, but breaks in the event that SOLVE makes more than 2^n queries to V^* ; in this case, S uses brute-force search to compute a witness $w \in \mathbf{R}_L(x)$, and continues in a straight-line simulation following the honest prover strategy. By linearity of expectation, the expected running time of S is

$$\begin{aligned} & \text{poly}(\mathbb{E}[\# \text{ queries made to } V^* \text{ by SOLVE }]) \\ & + \mathbb{E}[\text{time spent in straight-line simulation}] \end{aligned}$$

In Claim 19 below, we show that expected time spent in straight-line simulation is negligible. In Claim 20 below, we show that the expected number of queries made by SOLVE to V^* is at most $m^{2(d+1-\ell)}(2r)^{d+1-\ell}$. This two claims together implies Claim 18. □

Claim 19. *The expected time spent by S in straight-line simulation is negligible.*

Proof. The straight-line simulation takes at most $\text{poly}(2^n)$ steps since it takes $O(2^n)$ steps to extract a “fake” witness. Recall that, SOLVE runs the brute-force search only if it picks the same challenge (β) twice. Since, SOLVE is cut-off after 2^n steps, it can pick at most 2^n challenges. Therefore, by the union bound, the

probability that it obtains the same challenge twice is at most $\frac{2^n}{2^{n^2}}$. Thus, the expected time spent by S in straight-line simulation is at most $\frac{2^n}{2^{n^2}} \text{poly}(2^n)$, which is negligible. \square

Claim 20. *For all $x \in L, h, s, W, R$, $\ell \leq d$ such that $\text{SOLVE}_d^{V^*}(x, \ell, h, s, W, R)$ never outputs fails, $\mathbb{E}[\# \text{ queries by } \text{SOLVE}_d^{V^*}(x, \ell, h, s, W, R)] \leq m^{2(d+1-\ell)}(2r)^{d+1-\ell}$.*

Proof. We prove the claim by induction on ℓ . To simplify notation let $\alpha(\ell) = m^{2(d+1-\ell)}(2r)^{d+1-\ell}$. When $\ell = d + 1$, by design SOLVE does not perform any recursive calls or rewinds and therefore the number of queries made by SOLVE is bounded by the total number of prover messages (in a straight-line execution): $\text{poly}(mr)$ (m sessions of protocols with $\text{poly}(r)$ rounds).

Assume the claim is true for $\ell = \ell' + 1$. We show that it holds also for $\ell = \ell'$. Consider some fixed $x \in L, h, s, W, R$ such that $\text{SOLVE}_d^{V^*}(x, \ell', h, s, W, R)$ never outputs fails. We show that

$$\begin{aligned} \mathbb{E}[\# \text{ queries by } \text{SOLVE}_d^{V^*}(x, \ell', h, s, W, R)] &\leq m^{2(d+1-\ell')} r^{d+1-\ell'} \\ &= \alpha(\ell') = m^2(2r)\alpha(\ell' + 1) \end{aligned}$$

Towards this goal we introduce some additional notation. Given a view \hat{h} extending the view h ,

- Let $q_{\hat{s}}^{\ell'}(\hat{h})$ denote the probability that the view \hat{h} occurs in the “main-line” execution of $\text{SOLVE}_d^{V^*}(x, \ell', h, s, W, R)$ (i.e., starting on level ℓ') and that slot \hat{s} opens immediately after \hat{h} .
- Let $\Gamma_{\hat{s}}$ denote the set of views such that $q_{\hat{s}}^{\ell'}(\hat{h}) > 0$.

We bound the number of queries made by $\text{SOLVE}_d^{V^*}(x, \ell', h, s, W, R)$ as the sum of the queries SOLVE makes on level ℓ' , and the queries made by recursive

calls. The number of queries made by SOLVE on level ℓ' is upper-bounded by the total number of messages prover messages in a straight-line execution, i.e., $\text{poly}(mr)$. The number of queries made on recursive calls is computed by summing the queries made by recursive calls on over every slot \hat{s} and taking expectation over every view \hat{h} (such that $q_s^{\ell'}(\hat{h}) > 0$).

More precisely,

$$\mathbb{E}[\# \text{ queries by SOLVE}_d^{V^*}(x, \ell', h, s, \mathbf{W}, \mathbf{R})] \leq mr + \sum_{\hat{s}} \sum_{\hat{h} \in \Gamma_{\hat{s}}} q_s^{\ell'}(\hat{h}) E_{\hat{s}}(\hat{h})$$

where $E_{\hat{s}}(\hat{h})$ denotes the expected number of queries made by SOLVE from the view \hat{h} on \hat{s} . There are two steps involved in computing $E_{\hat{s}}(\hat{h})$. The first step involves finding the expected number of times SOLVE is run on a slot, and the second step using the induction hypothesis computing a bound for $E_{\hat{s}}(\hat{h})$.

Step 1: Given a view \hat{h} from where slot \hat{s} opens, let p^ℓ denote the probability that SOLVE rewinds slot \hat{s} from \hat{h} , i.e., p^ℓ is the probability that in the simulation from \hat{h} at level ℓ , V^* completes slot \hat{s} within time $g^{d-\ell}(1)$ from the opening of slot \hat{s} . Let y^ℓ denote the probability that when executing SOLVE at level ℓ from \hat{h} , V^* either aborts in the session of slot \hat{s} or takes more than time $g^{d-\ell}(\delta)$ to respond to slot \hat{s} . We clearly have that $p^\ell \leq 1 - y^\ell$ (note that equality does not necessarily hold since SOLVE might also return to a lower recursive level). Furthermore, it holds that $y^\ell = y^{\ell+1}$. This follows since SOLVE generates random Stage 1 messages, and uses the same (real) witness to generate Stage 2 messages, independent of the level of the recursion; additionally by Claim 14, SOLVE never halts outputting fail, we conclude that the view of V^* in the “main-line” simulation by SOLVE on level l is identically distributed to its view on level $l + 1$.

Therefore, the expected number of times SOLVE recursively executes \hat{s} at

level $\ell + 1$, before obtaining a good view, is at most $\frac{1}{1-y^{\ell+1}} = \frac{1}{1-y^\ell} \leq \frac{1}{p^\ell}$. Using linearity of expectation, the expected number of times SOLVE executes \hat{s} before obtaining m good views is at most $\frac{m}{p^\ell}$. Since, SOLVE rewinds \hat{s} from \hat{h} only with probability p^ℓ , the expected number of recursive calls to level $\ell + 1$ from \hat{h} is at most $p^\ell \frac{m}{p^\ell} = m$.

Step 2: From the induction hypothesis, we know that the expected number of queries made by SOLVE at level $\ell' + 1$ is at most $\alpha(\ell' + 1)$. Therefore, if SOLVE is run u times on a slot, the expected total number of queries made by SOLVE is bounded by $u\alpha(\ell' + 1)$. We conclude that

$$\begin{aligned} E_{\hat{s}}(\hat{h}) &\leq \sum_{u \in \mathbb{N}} \Pr[u \text{ recursive calls are made by SOLVE from } \hat{h}] u \alpha(\ell' + 1) \\ &= \alpha(\ell' + 1) \sum_{u \in \mathbb{N}} u \cdot \Pr[u \text{ recursive calls are made by SOLVE from } \hat{h}] \\ &\leq m \alpha(\ell' + 1) \end{aligned}$$

Therefore, $\mathbb{E}[\# \text{ queries by SOLVE}_d^{V^*}(x, \ell', h, s, \mathbf{W}, \mathbf{R})] \leq$

$$\begin{aligned} mr + \sum_{\hat{s}} \sum_{\hat{h} \in \Gamma_{\hat{s}}} q_{\hat{s}}^{\ell'}(\hat{h}) E_{\hat{s}}(\hat{h}) &\leq mr + \sum_{\hat{s}} m \alpha(\ell' + 1) \sum_{\hat{h} \in \Gamma_{\hat{s}}} q_{\hat{s}}^{\ell'}(\hat{h}) \\ &\leq mr + \sum_{\hat{s}} m \alpha(\ell' + 1) \leq mr + (mr) m \alpha(\ell' + 1) \leq \alpha(\ell') \end{aligned}$$

This completes the induction step and concludes the proof of Claim 20. \square

CHAPTER 4

ON THE COMPOSITION OF PUBLIC-COIN ZERO KNOWLEDGE PROTOCOLS

In this chapter we investigate a curious phenomenon regarding ZK and its composition. Three basic notions of compositions are sequential composition [31, 29], parallel composition [23, 27] and concurrent composition [23, 20]. In a sequential composition, the players sequentially run many instances of a zero-knowledge protocol, one after the other. In a parallel composition, the instances instead proceed in parallel, at the same pace. Finally, in a concurrent composition, messages from different instances of the protocol may be arbitrarily interleaved.

While the definition of ZK is closed under sequential composition [29], this no longer holds for parallel composition [27] (and thus not for concurrent composition either). As we have seen though, there are $\tilde{O}(\log n)$ -round black-box zero-knowledge protocols for all of NP that are secure under concurrent composition [64, 42, 61]. Moreover, if we require security to hold only under parallel composition, constant-round ZK protocols are known [25, 23, 26].

Whereas the original ZK protocols of [31, 28, 10] are *public-coin*—i.e., the verifier’s messages are its random coin-tosses—all of the aforementioned parallel or concurrent ZK protocols use *private coins*. Indeed, in their seminal paper, Goldreich and Krawczyk [27] show that only languages in BPP have *constant-round* public-coin (stand-alone) black-box ZK protocols with negligible soundness error, let alone the question of parallel composition. In particular, their results imply that (unless $\text{NP} \subseteq \text{BPP}$) the constant-round ZK protocols of e.g., [28, 10] with constant soundness error cannot be black-box ZK under parallel repetition (as

this would yield a constant-round black-box ZK protocol with negligible soundness error).

A natural question is whether the constant-round restriction imposed by the [27] result is necessary. Namely,

Is there a (possibly super-constant round) public-coin black-box ZK protocol that is secure under parallel (or even concurrent) composition?

4.1 Overview

In this chapter we provide a negative answer to the above question. Namely, we show that only languages in BPP have public-coin black-box ZK protocols that remain secure under parallel (and thus also concurrent) composition, regardless of round complexity.

Theorem 21. *If L has a public-coin argument that is black-box zero-knowledge and secure under parallel composition, then $L \in \text{BPP}$.*

In fact, our result establishes that any public-coin, black-box ZK protocol for a non-trivial language that remains secure under m parallel executions must have $\tilde{\Omega}(m^{1/2})$ rounds.

Perhaps our requirements are too stringent? Barak [3] is able to construct constant-round public-coin ZK arguments using non-black-box techniques, and achieve *bounded parallel security*. Indeed, we show that either one of two relaxations is sufficient to construct public-coin ZK protocols secure under (some form of) parallel composition; that is, we can construct *non-black-box* ZK ar-

guments secure against *full parallel composition*, or *black-box ZK* proofs secure against *bounded parallel composition*:¹

Theorem 22. *Assume the existence of one-way functions. Then for every polynomial m , there exists an $O(m^3)$ -round public-coin black-box zero-knowledge proof for NP that is secure under m -bounded concurrent composition.*

Theorem 23. *Assume the existence of collision-resistant hash-functions. Then there exists a constant-round public-coin parallel zero-knowledge argument for NP.*

Together, these results paint a full picture on the existence of public-coin ZK protocols secure against parallel repetitions, given the choice of black-box simulation vs. non-black-box simulation, and full concurrency vs. bounded concurrency.

Finally, we briefly turn to compositions in models with trusted set-up. Canetti, Goldreich, Goldwasser and Micali [13] show that in the Bare Public-Key (BPK) Model, where each player has a registered public-key, constant-round black-box concurrent ZK protocols exist for all of NP (whereas in the plain model without set-up, $\tilde{\Omega}(\log n)$ rounds are necessary for non-trivial languages [14]). We show that for the case of public-coin protocols, the BPK setup does not help with composition.

Theorem 24. *If L has a public-coin argument in the BPK model that is black-box zero-knowledge and secure under parallel composition, then $L \in \text{BPP}$.*

We remark that our lower bound does not extend to more elaborate public-key setups. For example, Damgrd [18] shows that a public key infrastructure

¹In fact, both Barak's construction and our black-box construction achieves bounded *concurrent* security.

with a certification authority can be used to construct constant-round public-coin arguments that are black-box concurrent zero-knowledge.

As we will see, some of the intermediate ideas in our work are closely related to the notion of *resettable soundness* [5]. Very informally, we establish that parallel repetition of public-coin protocols not only reduces the soundness error [58, 36], but also *qualitatively* strengthens the soundness—roughly speaking, the new protocols will be secure under a “resetting” attack.

4.1.1 Techniques for Theorem 21

To describe our techniques, first recall the Goldreich-Krawczyk [27] lower bound that only languages in **BPP** have $O(1)$ -round public-coin black-box ZK protocols. Let $\Pi = (P, V)$ be a public-coin black-box ZK protocol for a language L , and consider an adversarial verifier V^* that, instead of picking its messages at random, computes them by applying a hash function to the current transcript. [27] shows that any black-box simulator S , together with V^* , can decide L : on input x , simply run $S^{V^*}(x)$ and accept if S outputs an accepting view of V^* . Using the zero-knowledge property of Π , if $x \in L$, then $S^{V^*}(x)$ will output an accepting view of V^* (because an honest prover would convince V^*). The crux of their proof is then to show that if $x \notin L$, then $S^{V^*}(x)$ will not output an accepting view. If S does not rewind V^* , this would directly follow from the soundness of Π . However, S may rewind V^* , and may only convince V^* in one of its rewinding “threads”. Nonetheless, [27] manages to show that if S , by rewinding or “resetting” V^* , manages to trick V^* into accepting $x \notin L$, then we can construct a machine T (based on S) that manages to convince an external verifier V (with-

out rewinding V), contradicting the soundness of the protocol. In other words, they show that any $O(1)$ -round public-coin protocol is sound under a resetting-attack [13, 5], where the statement is fixed and the prover (simulator) running time is bounded by a fixed polynomial. Analogously, to prove our results, we show that any public-coin interactive protocol, repeated sufficiently many times in parallel, (and again letting the verifier pick its messages by applying a hash function to the transcript), is sound under a resetting-attack.

Previous reductions. The work of [27], as well as all subsequent black-box lower bounds (e.g., [43, 65, 14, 6, 41, 34]) relies on the following approach for constructing the stand-alone (non-resetting) prover T , given the rewinding simulator S . T incorporates S and internally emulates an execution of S with an internally emulated verifier (which of course can be rewound). During the emulation, T appropriately picks some messages sent by S to the internal verifier, and forwards them to an external verifier (and also forwards back the responses). The crux of the various lower bounds lies in choosing the externally forwarded messages so that the external verifier is convinced. The difficulty of this task stems from the fact that, at the time of deciding whether to externally forward a message or not, T does not yet know if S will eventually choose this message to “continue” its simulation (and use it as part of the output view), or treat this message simply as a “rewinding” (used to collect information).

For the case of constant-round protocols, [27] shows that externally forwarding a *random* selection of messages works; if the protocol has d rounds, this random selection is “correct” with probability at least $1/q^d$, where q is the number of queries made by the simulator to the verifier. This approach of simply running the simulator S “straight-line” seems hard to extend to protocols with a

polynomial number of rounds; the number of possible choices for messages to forward to the external verifier becomes too large.²

Our reduction. In our work, we are given a zero-knowledge protocol $\Pi = (P, V)$ for a language L that is secure under parallel repetitions. Building on the same framework as [27], we let V^{m*} be a verifier that starts m parallel sessions and generates its messages using hash-functions, let S be the black-box zero-knowledge simulator, and use $S^{V^{m*}}$ to decide L . As we will see, we choose the number of parallel sessions, m , as a (polynomial) function of the number of rounds in Π . Following the same argument, it is enough to show that on input $x \notin L$, S cannot produce an accepting view of V^{m*} . Because we may view S as a rewinding/resetting prover, it is equivalent to show that protocol (P^m, V^{m*}) is sound under resetting attacks. In the rest of this section we omit the common input x .

The crux of our work is then the following reduction: Given S , a *resetting* cheating prover of the *parallelized* protocol that convinces V^{m*} , we show how to construct T , a *straight-line* (non-rewinding) cheating prover of the original *single session* protocol that convinces V ; this contradicts the soundness of protocol Π . To further clarify the difference between S and T , let us compare the transcripts of an interaction between T and V , and of an interaction between S and V^{m*} . A transcript of the interaction between T and V is simply a transcript of a *single session* of the protocol Π ; each query from T to V is simply a prefix of the tran-

²For the case of sub-logarithmic-round protocols, Canetti, Kilian, Petrank and Rosen [14] show that when given the freedom to construct a concurrent adversarial verifier that can schedule messages in an arbitrary way, there exists some particular scheduling which makes it easy to identify appropriate messages to forward externally. Their work has the advantage that it applies to private-coin zero-knowledge protocols, but is not applicable in our setting due to the use of *concurrent* adversarial verifiers, and being limited to *sub-logarithmic-round* protocols. Incidentally, they also run the simulator S in a straight-line manner.

script that extends the previous query by one round of the protocol. A transcript of the interaction between S and V^{m*} can be much longer due to rewinds; furthermore, each query from S to V^{m*} is a prefix of a transcript of the *parallelized* protocol.

On a high level, T internally runs S with an internally simulated V^{m*} , and externally interacts with an external verifier V . In order to take advantage of S to convince the external verifier V , T “embeds” the interaction with V into the interaction between S and V^{m*} . This “embedding” is not straightforward for the following two reasons. Firstly, just as in [27], the external verifier V cannot be reset, whereas S may reset V^{m*} many times (i.e., S can make many more queries than the number of rounds of the protocol); as we will explain shortly, T carefully picks a subset of the rewindings to forward externally. Secondly, recall that V is a single session verifier, whereas V^{m*} is a m -session parallel verifier (looking forward, the reason we let V be a single session verifier is to enable T to appropriately pick which rewindings to forward). Therefore, T embeds the interaction with V only into a *single session* i of the m parallel sessions in the interaction between S and V^{m*} ; in fact, session i is picked uniformly random at the beginning and *fixed* throughout the execution of the reduction (looking forward again, the fact that session i is picked uniformly will be important for our analysis).

To summarize, T only externally forwards a subset of the S queries, and only forwards component i (corresponding to session i) of those queries. T then forwards back external responses from V as component i of the same subset of V^{m*} responses; all other V^{m*} responses are picked uniformly at random by T internally (this includes all except component i in the responses to the selected

subset of S queries, and all components of the remaining responses). Here we rely on the fact that Π is public-coin in order for T to generate V^{m*} responses in the forwarded session, despite the fact that other verifier responses in the forwarded session may be externally generated by V .

Recall that the difficulty of the reduction comes from choosing which S queries to forward externally. As remarked earlier, the approach of running S in a straight-line manner seems unlikely to work for polynomial-round protocols. Instead, we let T *rewind* S (while S itself believes it is rewinding the internally simulated V^{m*}). Our strategy is twofold. Firstly, T only externally forwards (component i of) queries that have a good chance of being included by S in its output (by assumption, S outputs a sequence of queries that convinces V^{m*}); because the protocol is public-coin, we can estimate this chance by doing internal test-runs. Secondly, once we have forwarded (component i of) a query, we “force” S to include the query in its output by repeatedly rewinding S while re-picking the internally generated V^{m*} messages (thus skewing the distribution of the internally generated V^{m*} messages).

To analyze T , we need to show that S would successfully convince the internally simulated V^{m*} , even though T has embedded the external interaction with V into the interaction between S and V^{m*} . Note that the success probability of S depends only on two inputs: the internally simulated V^{m*} messages, and the embedded external V messages (these can be found only in the forwarded session i). These two types of messages differ in that the internally simulated V^{m*} messages are picked by T , through the help of test-runs, to be “good”, while the external V messages are just uniform samples. We first show that if T is also allowed to rewind the external verifier V (which we cannot), ensuring that inter-

nal V^{m*} messages and external V responses are both “good”, then T only needs to perform polynomially many rewinds in order for S to successfully convince V^{m*} . Next, to remove the assumption of rewinding V , we use a probabilistic lemma due to Raz [63], originally used to prove that parallel repetition reduces the soundness error in two-prover games. We show that if there are enough parallel sessions, then not being able to pick “good” verifier responses in just one *random* session only introduces a small statistical error; since session i is picked uniformly at random at the beginning, this suffices for bounding the success probability of T .

ZK lower bounds and soundness amplification. As an independent contribution, we believe that our techniques elucidate an intriguing (and useful) connection between *lower bounds* for black-box ZK, and feasibility results for soundness/hardness amplification. Our techniques share many similarities with works on soundness amplification under parallel repetitions, such as [9, 58, 38], and especially [36]; in particular, our use of Raz’s lemma is similar to its use in [36]. Whereas those works show how to transform a parallel prover with “small” success probability into a stand-alone prover with “high” success probability, we have adapted their techniques to transform a *rewinding/resetting* parallel prover into a *non-rewinding* stand-alone prover.

As a further example of this connection, we extend our lower bound to the BPK model by relying again on techniques developed for soundness amplification. In the BPK model, we have the additional problem that the external verifier can decide whether to accept or reject based on its secret key, which T does not know. Consequently, T cannot determine whether the external verifier would accept or reject when doing test-runs, which is crucial for deciding which mes-

sages to forward externally. By relying on the “trust-halving” technique from [39, 9], and its refinement in [36], we show how T can make “educated guesses” on whether the external verifier accepts or not.

Extension to resettable soundness. More generally, the above techniques show how to transform a public-coin protocol so that it is sound under a weak form of resetting attack: where the statement is fixed, and the number of resets is a-priori bounded. Simply take a public-coin protocol, sufficiently repeat it in parallel, and let the verifier generate its messages by applying hash-functions to the current transcript. If the verifier uses pseudo-random functions instead of hash-functions as in [5], then we may remove the a-priori bound on the number of resets. Additionally, we show that if the original protocol is also a *proof of knowledge* [31, 23, 4], then the parallelized version satisfies the original (strongest) notion of resettable-soundness from [5], where the adversarial prover can also change the statement between resets. [5] showed a similar type of result for $O(1)$ -round public-coin proofs of knowledge.

4.1.2 Techniques for Theorem 22

Our public-coin black-box ZK protocol is similar in spirit to the concurrent zero-knowledge protocol of [64]. The proof has two stages. Stage one of the protocol contains multiple slots; in slot i , the prover commits to a random bit p_i using a statistically binding commitment, and the verifier replies with its own random bit v_i (note that unlike [64], the verifier does not commit to the bits v_i). Stage 2 is a modified witness indistinguishable proof of either the original statement, or that $p_i = v_i$ for “many” slots.

The intuition is that if the number of slots is sufficiently larger than the number of concurrent composition, then a simulator, with the aid of rewinds, can always force (i.e., rewind until) $p_i = v_i$ for “many” slots.

4.1.3 Techniques for Theorem 23

Let us briefly recall the idea behind Barak’s protocol (following a slight variant of this protocol due to [53]). Roughly speaking, for language L and common input $x \in \{0, 1\}^n$, the prover P and verifier V proceed in two stages. In Stage 1, P starts by sending a computationally-binding commitment $c \in \{0, 1\}^n$ to 0; V follows by sending a “challenge” $r \in \{0, 1\}^{2^n}$. In Stage 2, P shows (using a witness indistinguishable argument of knowledge) that either there exists a “short” string $s \in \{0, 1\}^n$ such that c is a commitment to a program Π such that $\Pi(s) = r$, or $x \in L$. Soundness follows from the fact that even if a malicious prover P^* tries to commit to some program Π (instead of committing to 0), with high probability, the string r sent by V will be different from $\Pi(s)$ for every string $s \in \{0, 1\}^n$. To prove ZK, consider the non-black-box simulator that commits to the code of the malicious verifier V^* ; note that by definition it holds that $\Pi(c) = V^*(c) = r$, and the simulator can use $s = c$ as a “fake” witness in the final proof.

Now, let us consider parallel composition. That is, we need to simulate the view of a verifier that starts $m = \text{poly}(n)$ parallel executions of the protocol. The above simulator no longer works in this setting: the problem is that the verifier’s code is now a function of *all* the commitments $\vec{c} = c_1, \dots, c_m$ sent in the different executions. (Note that if we increase the length of r , and therefore the allowed length of s , we can handle a bounded number of execution, by simply letting

$s = \vec{c}$). To get around this problem, we change the proof in the last stage as follows. Instead of proving the existence of a string s such that $\Pi(s) = r$, we show the existence of a seed $s \in \{0, 1\}^n$ for a pseudorandom function f , and an index $i \in \{0, 1\}^{\log^2 n}$, such that $\Pi_i(\vec{c}) = r$, where c_i is a commitment to Π using $f_s(i)$ as randomness, and $\Pi_i(x)$ is the “projection” of Π onto the i^{th} “coordinate”—i.e., the output of Π in the i^{th} parallel execution. To construct the zero-knowledge simulator, we start by picking a seed s , compute commitments c_i to Π using $f_s(i)$ as randomness, and use s and i as a “fake-witness” to simulate Stage 2 in execution i .

Outline

We give some preliminaries in Section 4.2, and jump into our impossibility results in Section 4.3 (standard model) and Section 4.4 (bare-public-key model). We then present our public-coin black-box bounded-concurrent zero-knowledge protocol in Section 4.5, and our public-coin non-black-box parallel zero-knowledge protocol in Section 4.6. Details of our application to resettable soundness can be found in Section 4.7.

4.2 Preliminaries

To aid our impossibility result, we recall the definition of resettable soundness. To aid our construction of non-black-box parallel zero-knowledge protocol, we recall the definition of universal arguments.

4.2.1 Resettable Soundness

Informally, given a protocol $\Pi = (P, V)$, a cheating prover P^* performing a **resetting attack** has the power to reset (i.e., rewind) the honest **resettable** verifier, resulting in multiple **sessions** of Π . Furthermore, in all these sessions, V uses the same random tape that is uniformly chosen before the attack. For example, a black-box zero-knowledge simulator is a valid resetting attack. We can consider two different models on how the input instances are chosen for each session. In the model of **resettable-soundness** as defined by [5], P^* can adaptively choose different input instances for each session. We also consider the model where P^* is given an input instance that must be used in all sessions (similar to the definition of resettable zero-knowledge by [13]); we call this **fixed-input resettable-soundness**.

Definition 23 (Resetting-Attack [5, Definition 3.1]). A **resetting attack** of a cheating prover P^* on a **resettable verifier** V is defined by the following two-step random process, indexed by a security parameter n :

1. Uniformly select and fix $t = \text{poly}(n)$ random-tapes, denoted r_1, \dots, r_t , for V , resulting in deterministic strategies V_{r_j} . When an input $x \in \{0, 1\}^n$ is also chosen, we call $V_{r_j}(x)$ an **incarnation** of V (i.e., V with its randomness set to r_j and common input set fixed to x).
2. On input 1^n , P^* is allowed to interact with $\text{poly}(n)$ incarnations of V . P^* chooses each incarnation (adaptively) by choosing $x \in \{0, 1\}^n$ and $j \in [t]$ (these choices may depend on P^* 's previous interactions with other incarnations of V). P^* may freely switch among interactions with different incarnations of V , and may rewind/reset each incarnation of V .

We further define two variants of resetting attacks. In a **fixed-input resetting attack**, the cheating prover P^* is given a fixed input instance x to use in all sessions. In a **q -query resetting attack**, the cheating prover P^* is allowed q queries total for verifier messages (summed over all interactions among the different incarnations of V).

Remark. We have chosen the “interleaving” attack model instead of the “non-interleaving” attack model, where P^* must finish its current interaction with an incarnation of V completely, before starting another interaction (see discussions in [13, 5]). The two models are equivalent as shown in [13]. We choose the “interleaving” model because later we will make the assumption that P^* never makes the same query twice to V . The notion of a q -query resetting attack is also more natural in the “interleaving” model.

Definition 24 (Resettable-Soundness [5, Definition 3.1]). Let $\Pi = (P, V)$ be a pair of interactive machines where V is PPT. We say Π is a **resettablely-sound proof** for a language L (resp., **resettablely-sound argument**) if the following condition holds:

Resettable-Soundness: For every resetting attack by P^* (resp., polynomial-size P^*), the probability that some incarnation $V_r(x)$ accepts and $x \notin L$ is negligible in n .

We say Π is a **q -query fixed-input resettablely-sound proof** (resp., **argument**) for a language L if the resettable-soundness property holds with respect to any q -query fixed-input resetting attack.

4.2.2 Universal Arguments

Universal arguments (introduced in [4] and closely related to CS-proofs [48]) are used in order to provide “efficient” proofs to statements of the form $y = (M, x, t)$, where y is considered to be a true statement if M is a non-deterministic machine that accepts x within t steps. The corresponding language and witness relation are denoted $L_{\mathcal{U}}$ and $\mathbf{R}_{\mathcal{U}}$ respectively, where the pair $((M, x, t), w)$ is in $\mathbf{R}_{\mathcal{U}}$ if M (viewed here as a two-input deterministic machine) accepts the pair (x, w) within t steps. Notice that every language in NP is linear time reducible to $L_{\mathcal{U}}$. Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all NP-statements. In fact, a proof system for $L_{\mathcal{U}}$ enables us to handle languages that are presumably “beyond” NP, as the language $L_{\mathcal{U}}$ is NE-complete (hence the name universal arguments).³

Definition 25 (Universal argument). A pair of interactive Turing machines (P, V) is called a **universal argument** system if it satisfies the following properties:

- **Efficient verification:** There exists a polynomial p such that for any $y = (M, x, t)$, the total time spent by the (probabilistic) verifier strategy V , on common input y , is at most $p(|y|)$. In particular, all messages exchanged in the protocol have length smaller than $p(|y|)$.
- **Completeness by a relatively efficient prover:** For every $((M, x, t), w)$ in $\mathbf{R}_{\mathcal{U}}$,

$$\Pr[(P(w), V)(M, x, t) = 1] = 1$$

Furthermore, there exists a polynomial q such that the total time spent by $P(w)$, on common input (M, x, t) , is at most $q(T_M(x, w)) \leq q(t)$, where $T_M(x, w)$ denotes the running time of M on input (x, w) .

³Furthermore, every language in NEXP is polynomial-time (but not linear-time) reducible to $L_{\mathcal{U}}$

- **Computational Soundness:** For every polynomial size circuit family $\{P_n^*\}_{n \in N}$, and every triplet $(M, x, t) \in \{0, 1\}^n \setminus L_{\mathcal{U}}$,

$$\Pr[(P_n^*, V)(M, x, t) = 1] < \nu(n)$$

where $\nu(\cdot)$ is a negligible function.

- **Weak proof of knowledge:** For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every polynomial-size circuit family $\{P_n^*\}_{n \in N}$, and every sufficiently long $y = (M, x, t) \in \{0, 1\}^*$ if $\Pr[(P_n^*, V)(y) = 1] > 1/p(|y|)$ then

$$\Pr[\exists w = w_1, \dots, w_t \in \mathbf{R}_{\mathcal{U}}(y) \text{ s.t. } \forall i \in [t], E_r^{P_n^*}(y, i) = w_i] > \frac{1}{p'(|y|)}$$

where $\mathbf{R}_{\mathcal{U}}(y) \stackrel{\text{def}}{=} \{w : (y, w) \in \mathbf{R}_{\mathcal{U}}\}$ and $E_r^{P_n^*}(\cdot, \cdot)$ denotes the function defined by fixing the random-tape of E to equal r , and providing the resulting E_r with oracle access to P_n^* .

4.3 Impossibility of Public-Coin Black-Box Parallel ZK

In this section we show that only languages in BPP have public-coin concurrent zero-knowledge protocols. We actually show a stronger result: Except for languages in BPP, no public-coin protocol remains black-box zero-knowledge when repeated in parallel. The formal theorems are stated below, where n denotes the security parameter or the input size.

Theorem 25. *Suppose language L has a $k = \text{poly}(n)$ -round public-coin black-box zero-knowledge proof Π with soundness error $1/2$. If $m \geq k \log^2 n$ and Π^m is zero-knowledge, then $L \in \text{BPP}$.*

Theorem 26. *Suppose language L has a $k = \text{poly}(n)$ -round public-coin black-box zero-knowledge argument Π with soundness error $1/2$. If $m \geq (k^2 \log k) \log^2 n$ and Π^m is zero-knowledge, then $L \in \text{BPP}$.*

The difference between Theorem 25 and 26 is caused by the difference between proofs and arguments. While the two theorems differ slightly in parameters, their proofs differ greatly. We remark that our theorems trivially hold with respect to “non-aborting” verifiers since we focus only on public-coin protocols.

4.3.1 Reducing to Resettable Soundness

The proofs of Theorem 25 and 26 begin in the same high-level framework as that of [27]. Suppose a language L has a k -round, public-coin ZK protocol $\Pi = (P, V)$, and Π^m is zero-knowledge with a black-box simulator S that runs in time n^d (as with all previous zero-knowledge lower bounds, we start by assuming that the simulator runs in *strict* polynomial time, and relax this assumption later in Section 4.3.4). To show that $L \in \text{BPP}$, we construct a “random-looking” adversarial verifier, V^* , and consider the following decision algorithm D : $D(x)$ runs $S^{V^*}(x)$ to generate a view of V^* , and accepts x if and only if V^* accepts given the generated view (which in turn occurs if and only if the honest verifier V accepts in all m sessions of the view).

V^* is actually a family of adversarial verifiers constructed as follows. Let H be a family of hash functions that is random enough compared to the running time of S ; formally, H should be n^d -wise independent (see [27, 15]). Given $h \leftarrow H$, let V_h^* be the verifier that when queried with transcript τ , responds (deterministically) with the message $h(\tau)$. We write $V^* = V_H^*$ to mean V_h^* for a

randomly chosen h , i.e., when D runs $S^{V_h^*}$, D first chooses h randomly from H and then run $S^{V_h^*}$.

We make two easy observations about S^{V^*} due to [27]. First, we may assume that whenever S queries V^* with a transcript or outputs a transcript τ , it first queries V^* with all the prefixes of τ ; this only increases the running time of S polynomially. Second, we may assume that S never queries V^* with the same transcript twice (instead S may keep a table of answers). Then the set of all responses generated by V_H^* is identical to the uniform distribution since H is n^d -independent and S makes at most n^d queries to V^* .

We need to show that decision procedure D is both complete and sound. Completeness states that if $x \in L$, then D should accept x with probability at least $2/3$. This easily follows: The output of $S^{V^*}(x)$ is indistinguishable from an interaction of (P^m, V^*) since S is a zero-knowledge simulator. Furthermore, (P^m, V^*) is identical to m copies of (P, V) since V^* produces independent, truly random verifier messages (made possible since V is public coin). Finally, by the completeness property of Π , V will accept x with probability 1 in all the copies of (P, V) .

Soundness states that if $x \notin L$, then D should accept with probability at most $1/3$. That is, $S^{V^*}(x)$ can produce an accepting view of V^* with probability at most $1/3$. Equivalently, we may view S as a n^d -query fixed-input resettable prover, and show that the protocol (P^m, V^*) is n^d -query fixed-input resettable sound. Therefore, Theorem 25 and 26 are completed by the following lemmas, respectively:

Lemma 27 (Resettable Sound Proofs). *Suppose $\Pi = (P, V)$ is a $k = \text{poly}(n)$ -round public-coin black-box zero-knowledge proof with soundness error $1/2$. If $m \geq k \log^2 n$*

and H is a family of $q = \text{poly}(n)$ -wise independent hash-functions, then (P^m, V_H^*) is q -query fixed-input resettably-sound.

Lemma 28 (Resettably Sound Arguments). *Suppose $\Pi = (P, V)$ is a $k = \text{poly}(n)$ -round public-coin black-box zero-knowledge argument with soundness error $1/2$. If $m \geq k^2 \log^2 n$ and H is a family of $q = \text{poly}(n)$ -wise independent hash-functions, then (P^m, V_H^*) is q -query fixed-input resettably-sound.*

Remark. Lemma 27 and 28 may be stronger than necessary in two ways. Firstly, the definition of resettable soundness requires negligible soundness error, while our main theorems only require soundness error $1/3$. Secondly, the definition of resettable soundness allows the resetting prover to interact with polynomially many copies of V_h^* with uniformly and independently chosen h 's, while the zero-knowledge simulator only interacts with one copy of V_h^* for a uniformly chosen h . This second difference is moot, however, because it is trivial to reduce a resetting attack on polynomially many copies of V_h^* (with uniformly and independently chosen h 's) to a resetting attack on a single copy of V_h^* (with uniformly chosen h), with only a polynomial loss in success probability. Therefore, in our proofs for Lemma 27 and 28, we only consider one copy of V_h^* .

4.3.2 Proof of Lemma 27: Resettably-Sound Proofs

Using the soundness amplification theorem of [2], protocol (P^m, V_H^*) has soundness error at most $1/2^m$. Let \hat{P}^* be a q -query fixed-input resettable prover. Suppose for the sake of contradiction that for some input $x \notin L$, V_H^* accepts a resettable interaction with \hat{P}^* with probability $1/p(n)$ for some polynomial p . We follow the strategy of [27] to use \hat{P}^* in order to break the soundness of (P^m, V_H^*) .

Whenever \hat{P}^* succeeds in breaking resettable soundness, \hat{P}^* would have queried V^* for k verifier messages that together form an accepting transcript of Π^m . A cheating prover of Π^m can therefore run \hat{P}^* internally, guess which queries of \hat{P}^* will form the accepting transcript, and forward them to an outside honest verifier of Π^m . Since \hat{P}^* queries V^* for at most $q(n)$ messages, the probability of guessing all the right queries is at least q^{-k} (one guess for each round of Π). Note that forwarding queries to an outside honest verifier does not lower the success probability of \hat{P}^* since V^* is identical to a honest verifier (they both respond with random messages). Thus this cheating prover, using \hat{P}^* , can break the soundness of Π^m with probability at least $(1/p)q^{-k} = 2^{-\Theta(k \log n)}$. Since $m \geq k \log^2 n$, we have $2^{-m} < 2^{-\Theta(k \log n)}$ and reach a contradiction. \square

4.3.3 Proof of Lemma 28: Resetably-Sound Arguments

We turn to prove our main result. Again we argue by contradiction. Suppose \hat{P}^* is a q -query fixed-input resettable prover, and suppose \hat{P}^* convinces V_H^* on some input $x \notin L$ with probability more than $1/p(n)$ for some polynomial p . We cannot repeat the proof of Lemma 27 because parallel repetitions cannot reduce the soundness of arguments beyond being negligibly small. Instead, we directly show a parallel repetition theorem for resettable soundness; that is, we relate the resettable soundness of (P^m, V_H^*) to the soundness of Π .

Proof Outline. The rest of this section describes how to construct a cheating prover T for Π . T runs \hat{P}^* internally and simulates V_H^* in response to \hat{P}^* queries. Every query made by \hat{P}^* is answered by a uniformly random reply. This perfectly simulates V_H^* since H is q -wise independent and \hat{P}^* makes at most

q queries (and never makes the same query twice); at the end of the q^{th} query, T will have implicitly defined a hash function $h \in H$ and simulated V_h^* , and \hat{P}^* will have successfully broken resettable soundness with probability $1/p(n)$ over the choice of these random replies (i.e., generated an accepting view of V_H^*).

To break the (stand-alone) soundness of Π , T chooses one of the m parallel sessions and forward a complete set of \hat{P}^* queries in that session (one for each round of Π) to an honest outside verifier V . The goal is to forward the queries on which \hat{P}^* is able to convince $V^* = V_H^*$ in protocol Π^m . This is challenging because \hat{P}^* may have multiple queries for each round of Π^m . While T must decide to forward a query or not at the time of the query, \hat{P}^* can wait until all queries are completed before choosing which queries to form an accepting view of V^* . To overcome this obstacle, a key part of our analysis relies on *rewinding* \hat{P}^* (note that at the same time, \hat{P}^* believes that it is rewinding V^*). Our strategy is twofold. First we only forward queries that has some chance (preferably a good chance) of being included a convincing transcript; this is done by doing test-runs of \hat{P}^* . Once we have forwarded a query, we force \hat{P}^* to use the query to convince V^* , by repeatedly rewinding \hat{P}^* .

We describe a **transcript** of \hat{P}^* as an alternating sequence of responses from T and queries from \hat{P}^* , $[t_1, s_1, t_2, s_2, \dots]$, where each \hat{P}^* -query s_i is in fact a partial transcript of Π^m that ends with a prover message, awaiting a verifier response. To avoid confusion, in our analysis, τ and $\langle \cdot \rangle$ denote views of V^* (transcripts of Π^m), while h and $[\cdot]$ denote transcripts of \hat{P}^* (transcripts of a resettable execution of Π^m). The goal of T is then to generate a full transcript h of \hat{P}^* in which \hat{P}^* generates a convincing transcript τ of (P^m, V^*) , while simultaneously having the foresight to forward (a session of) all the \hat{P}^* -queries pertaining to τ to the exter-

nal verifier V (i.e., all \hat{P}^* -queries in h that are a prefix of τ). If so, T has broken the soundness of Π , and we call this a **successful** simulation of \hat{P}^* . Note that because the randomness of \hat{P}^* is fixed, the behavior of \hat{P}^* is entirely determined by the T -responses in a transcript.

We start with a brief description of T . T first fixes a random session $\tilde{j} \in \{1, \dots, m\}$ to be forwarded. Then in k iterations (one for each round of Π), T incrementally fixes a transcript of \hat{P}^* and forwards a \hat{P}^* -query to V . In more details, at the beginning of iteration i , T starts with a partial transcript $h_i = [t_1, s_1, \dots, s_\ell]$ of \hat{P}^* that ends with $s_\ell = \tau_i$, a query for the i^{th} message of Π ($h_1 = []$, the empty transcript). Then:

Step 1. T forwards session \tilde{j} of the query τ_i to V , and receives a response $v_i^{(\tilde{j})}$.

Step 2. Fixing the reply $v_i^{(\tilde{j})}$, T uniformly samples completions of the partial transcript h_i until a “successful” completion h is found; specifically, \hat{P}^* on transcript h should produce an accepting view of V^* , τ , that extends the query τ_i . To move onto the next iteration, let τ_{i+1} be the length $i + 1$ prefix of τ , and let h_{i+1} be the prefix of h up until \hat{P}^* makes the query τ_{i+1} .

During the analysis, we first use Raz’s lemma to show that because the number of sessions is large and \tilde{j} was chosen randomly, we may pretend $v_i^{(\tilde{j})}$ is nicely chosen, conditioned on success, just like the other sessions (chosen by T in step 2). We also show that T rarely aborts.

Proof Details. We now introduce a series of hybrid simulators that formally defines T ; all our hybrids generate truly random responses to \hat{P}^* -queries so that \hat{P}^* cannot distinguish the hybrids from V^* . We start with a hypothetical hybrid,

and gradually move towards T .

Hybrid 1. Our first hybrid $T^{(1)}$ serves to introduce the general idea of how T queries \hat{P}^* internally; $T^{(1)}$ does not yet forward messages to the external verifier V .

$T^{(1)}$ builds a full transcript of \hat{P}^* in $k + 1$ iterations. In iteration i , $T^{(1)}$ fixes an \hat{P}^* -query τ_i for the i^{th} message of Π^m . This query should have a good chance of being used by \hat{P}^* in an accepting transcript of Π^m , and therefore is a good candidate to forward externally. Note that fixing an \hat{P}^* -query amounts to fixing the transcript of \hat{P}^* up until the desired \hat{P}^* -query is made.

We now describe $T^{(1)}$ in detail. In the very beginning, $T^{(1)}$ fixes a random session $\tilde{j} \in \{1, \dots, m\}$; eventually the \tilde{j}^{th} session will be forwarded externally. After that, $T^{(1)}$ incrementally grows a transcript of \hat{P}^* in k iterations. During the i^{th} iteration, $T^{(1)}$ receives a partial transcript of \hat{P}^* from the previous iteration, $h_i = [t_1, s_1, \dots, s_\ell = \tau_i]$, where τ_i is a \hat{P}^* -query for the i^{th} verifier message of Π^m ($h_1 = []$, the empty transcript). As an invariant maintained by $T^{(1)}$, it should be possible to extend h_i into a full transcript of \hat{P}^* where \hat{P}^* outputs an accepting view of V^* containing the query τ_i . We call such a full transcript a **successful completion** of h_i . Each iteration can be further divided into two steps:

Step 1. $T^{(1)}$ does not forward τ_i to the external V ; instead it simulates a response as follows. $T^{(1)}$ randomly samples a completion of h_i into h , conditioned on success (always possible due to the invariant). Let $v_i^{(\tilde{j})}$ be the response to τ_i in the \tilde{j}^{th} session in the successful completion h . Let \tilde{h}_i be a partial extension of the partial transcript h_i where the session \tilde{j} response to τ_i is fixed to $v_i^{(\tilde{j})}$ (but the responses in other sessions are not specified).

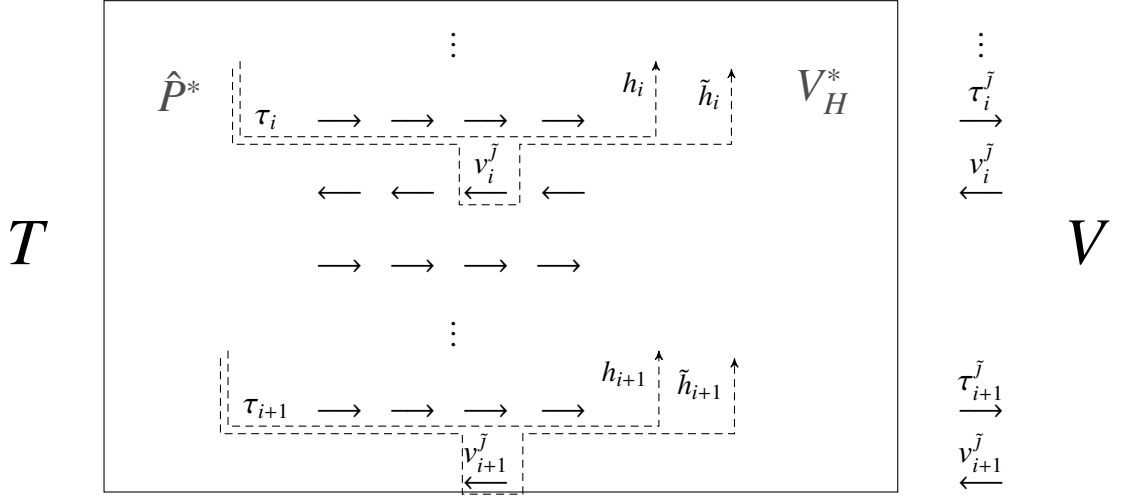


Figure 4.1: In order to interact with an outside honest verifier V , the reduction T internally maintains a partial interaction between the given resetting prover, \hat{P}^* , and the (supposedly resettably-sound) verifier V_H^* . The figure captures T after Step 1 of the $i + 1^{\text{st}}$ iteration, and illustrates some of the notations we define in the analysis.

Step 2. $T^{(1)}$ now samples a completion of \tilde{h}_i into \tilde{h} conditioned on success (note that h from the previous step is one such completion). Under transcript \tilde{h} , \hat{P}^* would output an accepting view τ of V^* (note that τ must extend τ_i). Let τ_{i+1} be the \hat{P}^* query for the $i + 1^{\text{st}}$ verifier message in τ (note that τ_{i+1} extends τ_i by definition of success). $T^{(1)}$ then sets h_{i+1} to be the prefix of \tilde{h} up to when \hat{P}^* makes the query τ_{i+1} . Note that the invariant holds since by definition \tilde{h} is a successful completion of h_{i+1} .

Note that in Step 2 of the final (k^{th}) iteration, $T^{(1)}$ simply outputs \tilde{h} as a full transcript of \hat{P}^* (there is no τ_{k+1} to fix). Due to the invariant, $T^{(1)}$ always produce a transcript of \hat{P}^* , on which \hat{P}^* outputs an accepting transcript τ . Moreover, the prefixes of τ would be the same τ_1, \dots, τ_k that were “chosen” by $T^{(1)}$ in each iteration (and would eventually be forwarded to the external verifier V in later hybrids).

Hybrid 2. Our second hybrid, $T^{(2)}$, describes a way to *efficiently* sample successful completions in Step 2 of each iteration (Step 1 will be replaced with the external verifier and is left alone for now). In Step 2, $T^{(2)}$ randomly completes the given partial execution (\tilde{h}_i) up to $100k^2pq$ times, until a successful completion is found. If none of the completions are successful, $T^{(2)}$ aborts. Note that conditioned on $T^{(2)}$ not aborting, the output distribution of $T^{(2)}$ is *identical* to $T^{(1)}$.

To show that $T^{(2)}$ aborts with small probability, suppose for now that $T^{(2)}$ is allowed to sample an unbounded number of completions. Let us bound the expected number of random completions that are needed to sample a successful one. In the following analysis we distinguish between two probability spaces: $\Pr_P[\cdot]$ is used to measure probabilities over a single execution of \hat{P}^* . On the other hand, $\Pr_T[\cdot]$ is used to measure probabilities over an execution of $T^{(2)}$ (with unbounded number of completions) which includes rewinding and executing \hat{P}^* multiple times.

Let H_i and \tilde{H}_i be the set of possible partial transcripts of \hat{P}^* that is given to $T^{(2)}$ in Step 1 and Step 2 of the i^{th} iteration, respectively. Given $h \in H_i$ (or \tilde{H}_i), let $\Pr_P[h]$ denote the probability that a transcript of \hat{P}^* has prefix h , and let $\Pr_T[h]$ denote the probability that $T^{(2)}$ is given h in the i^{th} iteration; similarly, $\Pr_P[\cdot \mid h]$ and $\Pr_T[\cdot \mid h]$ are probabilities conditioned on these events occurring. Let A^h be the event (over the \hat{P}^* probability space) that a transcript of \hat{P}^* has prefix h and is a successful completion of h ; as a special case, $A = A^\emptyset$ is just the event that \hat{P}^* outputs an accepting transcript. Also let R_i be the random variable (over the $T^{(2)}$ probability space) that denotes the number of completions performed by $T^{(2)}$ in step 2 of iteration i .

First we give a claim. Intuitively, the claim says that the probability of $T^{(2)}$

fixing h is proportional to the probability of successfully completing h ; the normalizing factor is simply $\Pr_P[A]$, the probability that \hat{P}^* produces an accepting transcript.

Claim 29. *Let $h \in \tilde{H}_i$. $\Pr_T[h] \Pr_P[A] = \Pr_P[A^h]$.*

Proof. Recall that the behavior of \hat{P}^* is entirely determined by the random messages generated by $T^{(2)}$. Let us consider a complete binary tree \mathcal{T} of depth n^d that represents all possible length n^d random bit-strings generated by $T^{(2)}$. Then every partial execution of \hat{P}^* corresponds to a node in \mathcal{T} based on the verifier messages received so far by \hat{P}^* in h .

Let us focus on the leaf nodes in \mathcal{T} since they occur with equal probability. Given h , define $L(h)$ to be the set of leaf nodes in \mathcal{T} that are children of h ; these nodes corresponds to possible completions of h . We also define $G(h)$ to be the subset of $L(h)$ that corresponds to successful completions of h (i.e. leaves where the event A^h is true). Finally let $L_0 = L(\emptyset)$ be all the leaf nodes, and $G_0 = G(\emptyset)$ be the subset of L_0 that corresponds to executions where \hat{P}^* produces an accepting transcript.

Recall that our goal is to prove that

$$\Pr_T[h] \Pr_P[A] = \Pr_P[A^h] .$$

Clearly

$$\Pr_P[A] = \frac{|G_0|}{|L_0|} \quad \Pr_P[A^h] = \frac{|G(h)|}{|L_0|} \tag{4.1}$$

To expand $\Pr_T[h]$, let $\tilde{h}_1, h_2, \dots, h_i, \tilde{h}_i = h$ be the prefixes of h given to $T^{(1)}$ in previous steps of previous iterations. As we see below, the expression for $\Pr_T[h]$

telescopes:

$$\begin{aligned}
\Pr_T[h] &= \Pr_T[\tilde{h}_1] \prod_{\ell=2}^i \Pr_T[h_\ell \mid \tilde{h}_{\ell-1}] \Pr_T[\tilde{h}_\ell \mid h_\ell] \\
&= \frac{|G(\tilde{h}_1)|}{|G_0|} \prod_{\ell=2}^i \frac{|G(h_\ell)|}{|G(\tilde{h}_{\ell-1})|} \frac{|G(\tilde{h}_\ell)|}{|G(h_\ell)|} \\
&= \frac{|G(\tilde{h}_i)|}{|G_0|} = \frac{|G(h)|}{|G_0|}
\end{aligned} \tag{4.2}$$

Equations (4.1) and (4.2) together gives the claim. \square

Now we bound the expected number of samples needed to find a successful completion.

Lemma 30. $\mathbb{E}_T[R_i] \leq pq$.

Proof. First expand $\mathbb{E}_T[R_i]$ by conditioning on the transcript h fixed in Step 1:

$$\mathbb{E}_T[R_i] = \sum_{h \in \tilde{H}_i} \Pr_T[h] \mathbb{E}_T[R_i \mid h] \tag{4.3}$$

Recall that in Step 2, $T^{(2)}$ samples random completions of h until a successful completion is found. Therefore

$$\mathbb{E}_T[R_i \mid h] = \frac{1}{\Pr_P[A^h \mid h]} \Rightarrow \mathbb{E}_T[R_i] = \sum_{h \in \tilde{H}_i} \Pr_T[h] \frac{1}{\Pr_P[A^h \mid h]} \tag{4.4}$$

By expanding the RHS of Claim 29 and rearranging terms, we have

$$\begin{aligned}
\Pr_T[h] \Pr_P[A] &= \Pr_P[A^h] = \Pr_P[h] \Pr_P[A^h \mid h] \\
\Rightarrow \Pr_T[h] \frac{1}{\Pr_P[A^h \mid h]} &= \Pr_P[h] \frac{1}{\Pr_P[A]} \leq p \Pr_P[h]
\end{aligned}$$

since we assumed $\Pr_P[A] \geq 1/p$. Substituting this back into (4.4) gives

$$\mathbb{E}_T[R_i] \leq p \sum_{h \in \tilde{H}_i} \Pr_P[h] \tag{4.5}$$

Finally, we may break up the set \tilde{H}_i based on the length of h which ranges from 1 to q (where length is the number of \hat{P}^* -queries). Since each transcript of \hat{P}^* has exactly one length ℓ prefix:

$$\mathbb{E}_T[R_i] \leq p \sum_{\ell=1}^q \sum_{h \in \tilde{H}_i, |h|=\ell} \Pr_P[h] \leq p \sum_{\ell=1}^q 1 = pq \quad \square$$

Finally, we show that $100k^2pq$ random completions are enough for $T^{(2)}$.

Lemma 31. $T^{(2)}$ aborts with probability at most $1/5$.

Proof. Since $\mathbb{E}_T[R_i] = \sum_{\tilde{h}_i} \Pr_T[\tilde{h}_i] \mathbb{E}_T[R_i \mid \tilde{h}_i] = \mathbb{E}_T[\mathbb{E}_T[R_i \mid \tilde{h}_i]] \leq pq$, the Markov inequality states that the probability of $T^{(2)}$ fixing an \tilde{h}_i such that $\mathbb{E}_T[R_i \mid \tilde{h}_i] \geq 10k pq$ is at most $1/(10k)$. For each “good” \tilde{h}_i where $\mathbb{E}_T[R_i \mid \tilde{h}_i] < 10k pq$, we apply the Markov inequality again to obtain $\Pr_T[R_i \geq 100k^2 pq \mid \tilde{h}_i] \leq 1/(10k)$. Using the union bound we see that in any iteration, $T^{(2)}$ aborts in Step 1 with probability at most $1/(5k)$. A final union bound over k iterations of Step 2 shows that $T^{(2)}$ aborts overall with probability at most $1/5$. \square

Hybrid 3. Our third and final hybrid $T^{(3)} = T$ differs from $T^{(2)}$ in Step 1 of each iteration. Recall that some session \tilde{j} is chosen randomly as the forwarding session. Instead of generating $v_i^{(\tilde{j})}$ in Step 1, $T^{(3)}$ asks the external honest verifier V for a verifier message. Because Π is public-coin, $T^{(3)}$ can continue to complete partial transcripts of \hat{P}^* even if session \tilde{j} is forwarded to V externally.

Given transcript $h_i = [t_1, s_1, \dots, s_\ell = \tau_i]$ in iteration i , $T^{(3)}$ forwards session \tilde{j} of τ_i to V , and uses the response from V as $v_i^{(\tilde{j})}$ in Step 2.⁴ Suppose for now that

⁴Strictly speaking, the interaction between $T^{(3)}$ and the honest verifier V is non-resetting. Therefore, instead of forwarding session \tilde{j} of query τ_i to V , $T^{(3)}$ simply sends the last prover message in session \tilde{j} of the query τ_i to V . For ease of exposition, we continue to use the phrase “ $T^{(3)}$ forwards the query τ_i ” to mean the above.

$T^{(3)}$ does not abort and terminates successfully. Then \hat{P}^* would have generated an accepting transcript τ of Π^m . Since τ_1, \dots, τ_k are prefixes of τ , session \tilde{j} of τ would be an accepting transcript of Π consisting of forwarded prover messages and responses from V . This breaks the soundness of Π .

Therefore, it remains to show that $T^{(3)}$ is successful with probability more than $1/2$. We will use Raz's lemma [63, Claim 5.1] in analogy with [38, 36] to show that $v_i^{(j)}$ as generated by $T^{(1)}$ and $T^{(2)}$ is actually very close to the uniformly random messages generated by the honest verifier V . First we cite Raz's lemma as it appears in [37, Lemma 5]:

Lemma 32. *Let $\{U_j\}_{j \in [m]}$ be independent random variables on \mathcal{U} with probability distribution P_{U_j} . Let W be an event in \mathcal{U}^m and $\Pr[W]$ be measured according to the joint probability distribution $\Pi_j P_{U_j}$. Then*

$$\sum_{j=1}^m \Delta(U_j|W, U_j) \leq \sqrt{m \log \left(\frac{1}{\Pr[W]} \right)}$$

where Δ is the statistical distance between distributions, and $U_j|W$ is the j^{th} component of an element in \mathcal{U}^m chosen based on the joint probability distribution $\Pi_j P_{U_j}$, conditioned on W .

In other words, let $\{U_j\}_j$ be independent random variables, and let W be an event over $\Pi_j U_j$. If W occurs with high probability and there are many U_j , then on average over j , sampling U_j conditioned on W does not differ much from simply sampling U_j . Lemma 32 allows us to bound the change in success probability when $T^{(3)}$ forwards messages from a random session to V .

Lemma 33. *$T^{(3)}$ fails with probability at most $3/10 + O(1/\log n)$.*

Proof. We first construct a series of finer hybrids, T_1, \dots, T_{k+1} , where T_i proceeds

as $T^{(2)}$ until the start of iteration i (no forwarding), and continues as $T^{(3)}$ afterwards (with forwarding)⁵. Observe that $T_1 = T^{(3)}$ and $T_{k+1} = T^{(2)}$.

Consider two neighboring hybrids, T_i and T_{i+1} , which differ only in iteration i . Let h be the partial execution given in iteration i . For $j \in [m]$, let U_j be the random variable that denotes all the additional session j messages sent by T to randomly complete h , i.e., $\{U_j\}_j$ are independent and uniformly random. Let W^h be the event that the random messages U_1, \dots, U_m together produced a successful completion of h . By definition, the distribution of $v_i^{(j)}$ produced by T_{i+1} (i.e., $T^{(2)}$) is just the first message of $U_j|W^h$. On the other hand, the distribution of $v_i^{(j)}$ produced by T_i (i.e., $T^{(3)}$) is just the uniform distribution, just like the first message of U_j .

Since T_{i-1} and T_i only differ in how $v_i^{(j)}$ is produced, their difference in success probability can be bounded by the statistical difference in the distributions of $v_i^{(j)}$. This is in turn bounded by:

$$\sum_{h \in H_i} \sum_{j=1}^m \Pr_T[h] \Pr[\tilde{j} = j] \Delta(U_j|W^h, U_j) = \sum_{h \in H_i} \Pr_T[h] \left(\frac{1}{m} \sum_{j=1}^m \Delta(U_j|W^h, U_j) \right) \quad (*)$$

Lemma 32 states that for any event W ,

$$\frac{1}{m} \sum_{j=1}^m \Delta(U_j|W, U_j) \leq \sqrt{\frac{1}{m} \log \left(\frac{1}{\Pr[W]} \right)}$$

Observe that before iteration i , T_i and T_{i+1} are identical to $T^{(2)}$. When $T^{(2)}$ does not abort, $T^{(2)}$ is identical to $T^{(1)}$. In that case, Lemma 30 along with the Markov inequality implies that except with probability $1/(10k)$, $T^{(2)}$ fixes a “good” h with $\mathbb{E}_T[R_i | h] \leq 10kpq$, so that

$$\Pr[W^h] = \Pr_P[A^h | h] = \frac{1}{\mathbb{E}_T[R_i | h]} \geq \frac{1}{10kpq}$$

⁵This still makes sense since Π is a public-coin protocol; the outside verifier can directly generate a verifier response for any round of the protocol.

We can now break the sum in (*) into two parts. Observe that

$$\sum_{\text{bad } h \in H_i} \Pr_T[h] \left(\frac{1}{m} \sum_{j=1}^m \Delta(U_j | W^h, U_j) \right) \leq \sum_{\text{bad } h \in H_i} \Pr_T[h] \leq \frac{1}{10k}$$

since statistical distances are upper bounded by 1, and

$$\begin{aligned} & \sum_{\text{good } h \in H_i} \Pr_T[h] \left(\frac{1}{m} \sum_{j=1}^m \Delta(U_j | W^h, U_j) \right) \\ & \leq \sum_{\text{good } h \in H_i} \Pr_T[h] \sqrt{\frac{1}{m} \log(10kpq)} \leq \sqrt{\frac{1}{m} \log(10kpq)} \end{aligned}$$

since $\sum_{h \in H_i} \Pr_T[h] = 1$. Together, they show that (*) is at most

$$\frac{1}{10k} + \sqrt{\frac{1}{m} \log(10kpq)} = \frac{1}{10k} + O\left(\frac{1}{k\sqrt{\log n}}\right)$$

since $m \geq k^2 \log^2 n$. Summing up over the hybrids, and recalling that $T^{(2)}$ fails with probability at most $1/5$ (Lemma 31), $T^{(3)}$ fails with probability at most

$$\frac{1}{5} + k \left(\frac{1}{10k} + O\left(\frac{1}{k\sqrt{\log n}}\right) \right) \leq \frac{3}{10} + O\left(\frac{1}{\sqrt{\log n}}\right)$$

□

Lemma 33 shows that T is successful with probability $> 1/2$, and completes the proof of Lemma 28.

4.3.4 On Expected Simulation

As with most lower bounds for black-box zero-knowledge, a careful reading reveals that Theorems 25 and 26 also apply to more liberal definitions of zero-knowledge, such as ε -zero-knowledge⁶ [20]. In particular, the theorems also apply to zero-knowledge with expected polynomial time simulators.

⁶In ε -zero-knowledge, the indistinguishability gap between the view of V^* and the view generated by the simulator is allowed to be some inverse polynomial approaching zero, as opposed to negligible.

First observe that expected time simulation (say with a polynomial expected running time $q(n)$) achieves ε -zero-knowledge with $\varepsilon = 1/q + \nu$ for some negligible function ν : simply by cut off the simulation after $(q(n))^2$ steps and apply the Markov inequality. Next observe the zero-knowledge property is only required to show that the decision process \mathcal{D} is complete for language L with probability $2/3$. In the case of ε -zero-knowledge, we simply need $\varepsilon < 1/3$ to hold, which is true for sufficiently large n .

4.4 Impossibility in the Bare Public Key Model

Many setup assumptions have been used to construct concurrent zero-knowledge with better efficiency than the standard model. For example, in the CRS (common reference string) model, even non-interactive zero-knowledge is possible [22]. Other “weaker” setups have produced varying results, and we will be concentrating on the *bare public key* model.

In the Bare Public-Key (BPK) model [13], every player has a public key that can be accessed by any other player. When a protocol is repeated in parallel, we assume that the honest parties use fresh independent public keys for each parallel session. By assuming that all public keys are properly registered before a protocol begins, Canetti, Goldreich, Goldwasser and Micali [13] showed that *constant-round*, private-coin arguments exist for NP even if we require black-box *resettable zero-knowledge*, a property that implies black-box concurrent zero-knowledge. In contrast, in the plain model, $\tilde{O}(\log n)$ rounds are required for concurrent black-box zero-knowledge proofs [14]. It is therefore natural to ask if the BPK setup can overcome our lowerbound for public-coin zero-knowledge

protocols.

In this section we extend our impossibility result from Sect. 4.3 to the BPK model. We actually extend our result to a larger class of **slightly-private-coin** protocols, defined with the following properties:

1. The first message of the protocol, from the verifier, is allowed to be private coin. All other subsequent verifier messages are public-coin, i.e., independent segments of the verifier's random tape.
2. At the end of the protocol, the verifier may run a private coin algorithm to accept or reject the interaction. In particular, the verifier's decision may depend on the private coins used to generate the first message.

Note that every public-coin protocol in the BPK model can be transformed into a slightly-private-coin protocol, because

1. The verifier can send its public key to the prover in the first message (property 1).
2. The verifier can base its acceptance decision on its secret key (property 2).

Our modified theorem is the following:

Theorem 34. *Suppose language L has a $k = \text{poly}(n)$ -round slightly-private-coin black-box zero-knowledge argument Π with negligible soundness error in n . If $m \geq (k^2 \log^2 k) \log^2 n$ and Π^m is zero-knowledge, then $L \in \text{BPP}$.*

Recall that in the analysis of Theorem 26, we treat the black-box zero-knowledge simulator S as a resetting prover \hat{P}^* of (P^m, V^*) , and use \hat{P}^* to construct a machine T , which in turn contradicts the soundness of Π . We now have a problem

whenever T needs to sample a successful completion of a partial transcript of \hat{P}^* , since T does not know whether the external verifier V would accept or reject the transcript produced by \hat{P}^* . To overcome this problem, we follow an approach similar to [9, 36] by guessing whether V would accept or reject based on whether the other verifiers, simulated by T , accept or reject their respective parallel sessions.

Proof. We extend the analysis of Theorem 26 in analogy with [36]. We first describe how T guesses if V accepts or rejects in the forwarded session \tilde{j} . Whenever T completes a partial execution of \hat{P}^* , let $z_{-\tilde{j}}$ be the number of sessions, excluding session \tilde{j} , in which S produced a rejecting view. We exclude session \tilde{j} for the aforementioned reason that without knowing the private key (or private coins) of the external verifier V , T cannot tell if V will accept or reject the view.

Let $w_{-\tilde{j}}$ be a Bernoulli random variable with $\Pr[w_{-\tilde{j}} = 1] = 2^{-\nu z_{-\tilde{j}}}$, where ν is an asymptotically small parameter to be determined later. $w_{-\tilde{j}}$ corresponds to T 's guess: If $w_{-\tilde{j}} = 1$, T will consider the completion successful, and vice versa. Intuitively, T is more likely to consider a completion as a success if the number of rejecting sessions is fewer.

To facilitate the analysis, we also consider a hypothetical but more symmetric process. Given a transcript generated by \hat{P}^* , let z be the number of sessions, including session \tilde{j} , in which \hat{P}^* produced a rejecting view. Similarly, let w be the Bernoulli random variable with $\Pr[w = 1] = 2^{-\nu z}$.

We now prove Theorem 34 with the same framework as Theorem 26, using the following modified hybrids. Hybrids $T^{(1)}$, $T^{(2)}$ and $T^{(3)}$ are constructed as before, except they now compute z and w to determine if a completion is suc-

cessful. The final machine, T , differs from $T^{(3)}$ by computing $z_{-\tilde{j}}$ and $w_{-\tilde{j}}$ instead.

Claim 35. *The probability that $T^{(1)}$ generates a rejecting view in session \tilde{j} is at most:*

$$\frac{3}{m} \left(\frac{-\log v^2}{v} + 4 \right)$$

Proof. The proof of this claim essentially follows from an analysis in [36] (which contained more general parameters). For the sake of completeness, we include their analysis without the extra parameters here.

Before introducing the public key extension, $T^{(1)}$ simply samples a random successful transcript of \hat{P}^* (see Claim 29). After adopting the new notion of success based on w , $T^{(1)}$ now samples a random transcript of \hat{P}^* conditioned on $w = 1$. That is, $T^{(1)}$ outputs a transcript of \hat{P}^* that generates rejecting views in j sessions with probability proportional to 2^{-vj} .

Since $T^{(1)}$ chooses \tilde{j} randomly, it is enough to bound the expected number of rejecting sessions. Let p_j be the probability that in a random execution of \hat{P}^* , the output view contains j rejecting sessions. Then, the expected number of rejecting verifiers is

$$\frac{\sum_{j=0}^m j p_j 2^{-vj}}{\sum_{j=0}^m p_j 2^{-vj}} \quad (4.6)$$

[36] gives a bound of (4.6) with more general parameters. For the sake of completeness, we include their analysis below without the extra parameters.

Recall that by assumption, \hat{P}^* generates an output view in which all sessions accept with probability at least $1/3$. Therefore we can lower bound the denominator of (4.6) by

$$\sum_{j=0}^m p_j 2^{-vj} \geq p_0 \geq 1/3 .$$

To upper bound the numerator, we use the following inequality:

$$\sum_{j=0}^{\infty} j2^{-vj} = \frac{2^{-v}}{(1 - 2^{-v})^2} \leq \frac{1}{(1 - 2^{-v})^2} \leq \frac{4}{v^2} .$$

The last inequality follow from the fact that $1 - 2^{-v} \geq v/2$ for small v . Directly apply this bound to the numerator (using $p_j \leq 1$) gives an overly loose bound since v is asymptotically small. Instead, we split the expression of the numerator at some parameter t :

$$\begin{aligned} \sum_{j=0}^m jp_j 2^{-vj} &\leq t \sum_{j=0}^m p_j 2^{-vj} + \sum_{j=1}^{m-t} jp_{t+j} 2^{-v(t+j)} \\ &\leq t + \frac{4}{v^2} 2^{-vt} . \end{aligned}$$

Setting $t = -\log v^2/v$, we see that the expected number of rejecting verifiers is at most

$$3 \left(\frac{-\log v^2}{v} + 4 \right) .$$

Since $T^{(1)}$ chooses \tilde{j} uniformly from $\{1, \dots, k\}$, the probability that $T^{(1)}$ outputs a view that rejects in session \tilde{j} is

$$\frac{3}{m} \left(\frac{-\log v^2}{v} + 4 \right) .$$

□

Lemma 36. *The probability that $T^{(2)}$ aborts is at most $1/5$. Otherwise, the output of $T^{(2)}$ is identical to $T^{(1)}$.*

Proof. By computing w and z , there are now more “successful” executions than before (originally, only executions where $z = 0$, i.e., no rejecting sessions, are successful). Therefore, $T^{(2)}$ now aborts with less probability than before, which is bounded by $1/5$ (Lemma 31). □

Lemma 37. $T^{(3)}$ fails to produce an accepting view in session \tilde{j} with probability at most

$$\frac{3}{m} \left(\frac{-\log v^2}{v} + 4 \right) + \frac{3}{10} + O\left(\frac{1}{\log n}\right)$$

Proof. This follows from Claim 35, and by applying Raz's lemma in the same manner as in Lemma 33. \square

Lemma 38. The output of $T^{(3)}$ and T differs statistically by at most kv .

Proof. $T^{(3)}$ and T differs in how a successful completion is recognized. For any completion, the difference in probability of it being considered successful by $T^{(3)}$ and T is:

$$\Pr[w_{-\tilde{j}} = 1] - \Pr[w = 1] = 2^{-vz_{-\tilde{j}}} - 2^{-vz} \leq 2^{-v(z-1)} - 2^{-vz} \leq 1 - 2^{-v} \leq v .$$

For each round of protocol Π , $T^{(3)}$ and T repeatedly perform the same task (completing partial transcript of S) until $w = 1$ or $w_{-\tilde{j}} = 1$, respectively. Therefore the statistical difference between the two process is at most kv . \square

Combining Lemma 37 and 38, we see that T fails to break the soundness of Π with probability at most

$$\frac{3}{m} \left(\frac{-\log v^2}{v} + 4 \right) + \frac{3}{10} + O\left(\frac{1}{\sqrt{\log n}}\right) + kv$$

By setting $v = 1/\sqrt{km}$, the expression becomes

$$3\sqrt{\frac{k}{m}} \log(km) + \frac{12}{m} + \frac{3}{10} + O\left(\frac{1}{\sqrt{\log n}}\right) + \sqrt{\frac{k}{m}}$$

Since $m \geq k^2 \log^2 k \log^2 n$, we conclude that T fails with probability at most $3/10 + o(1)$. That is, T succeeds with non-negligible probability, contradicting the soundness of Π . \square

4.5 Public-Coin Black-Box Bounded Concurrent ZK

In this section we give a family `BOUNDEDCONCZK` of public-coin proofs for `NP`, parametrized by k . The proof with parameter k has $2k^3 + 4$ rounds, and is k -bounded concurrent zero-knowledge assuming the existence of one-way functions, whenever $k = \omega(\log n)$ where n is the input size. `BOUNDEDCONCZK` requires the use of statistically binding commitment schemes.

4.5.1 The Protocol

Our construction of `BOUNDEDCONCZK` is similar in spirit to the concurrent zero-knowledge protocol of [64]. Given a language $L \in \text{NP}$ and a parameter k , we construct a two stage public-coin proof (P, V) as follows. In stage one, $2k^3$ rounds of messages are exchanged where in each round, the prover gives a statistically binding commitment of a random bit p_i , and the verifier responds with a random bit v_i ; we call $p_i = v_i$ a **correct guess** (note that unlike [64], the verifier does not commit to the bits v_i). In stage two, (P, V) runs a 4-round public-coin witness indistinguishable proof of the modified NP statement “either $x \in L$ or that $p_i = v_i$ for $k^3 + k^2/2$ values of i ”, where x is the problem instance. This can be instantiated with a parallel repetition of the Blum Hamiltonicity protocol [10] with 2-round statistically binding commitments constructed from one-way functions. The verifier accepts if the prover is successful with the stage two proof.

We choose $2k^3$ rounds of interaction in Stage One of `BOUNDEDCONCZK` for the following two reasons. First, by the Chernoff bound, we expect that no ad-

PROTOCOL BOUNDEDCONCZK

Common Input: An instance x of a language $L \in \text{NP}$ and a parameter k .

Stage One: For i from 1 to $2k^3$:

$P \rightarrow V$: Commit to a random bit p_i using a statistically binding commitment.

$V \rightarrow P$: Reply with a random bit v_i .

Stage Two: A 4-round public-coin witness indistinguishable proof (e.g., parallel repetitions of the Blum Hamiltonicity protocol [10]) of the NP statement:

$$\left(\text{there exist distinct } i_1, \dots, i_{k^3 + \frac{1}{2}k^2} \text{ s.t. } p_{i_j} = v_{i_j} \text{ for all } j \right) \vee (x \in L)$$

Figure 4.2: Our public-coin black-box bounded concurrent zero-knowledge protocol.

versarial prover can have more than $k^3 + O(\sqrt{k^3})$ correct guesses. Hence BOUNDEDCONCZK is sound. On the other hand, a zero-knowledge simulator can repeatedly rewind the verifier until it gets a correct guess. Intuitively (and shown formally later), in each round of stage one, the simulator can set one extra $p_i = v_i$ for *some* session, in addition to “natural luck” (that gives correct guesses for half of the sessions). Since the number of sessions is bounded by k , the simulator is able to have $k^3 + O(k^3/k) = k^3 + O(k^2)$ correct guesses per session. This provides the simulator with a trapdoor to simulate stage two of the protocol, and hence BOUNDEDCONCZK is bounded concurrent zero-knowledge. We remark that k^3 was chosen for the sake of simplicity and is not optimized. We show completeness and soundness below.

BOUNDEDCONCZK is clearly complete. A prover given a correct problem instance and witness pair, $(x \in L, w)$, can commit to random bits in stage one, and use w to successfully complete the stage two proof.

We next show that BOUNDEDCONCZK has negligible soundness error. Suppose $x \notin L$. Then there are two ways for the prover to mislead the verifier:

1. The prover may have $p_i = v_i$ for $k^3 + k^2/2$ (or more) values of i either by breaking the binding property of the commitment, or by guessing luckily. The former occurs with negligible probability since the commitment is statistically binding. The latter occurs with probability $e^{-k/4}$ by the Chernoff bound⁷.
2. Otherwise, the prover may break the soundness of the stage two proof, which occurs with probability at most 2^{-k} due to the parallel repetitions.

Since $k = \omega(\log n)$, both $e^{-k/4}$ and 2^{-k} are negligible in n .

4.5.2 Black-Box Bounded Concurrent ZK

We construct a black box simulator S such that given an adversarial verifier, V^* , S^{V^*} generates the view of V^* in BOUNDEDCONCZK, provided that the number of concurrent sessions m satisfies $m \leq k$. The goal of S is to obtain as many correct guesses as possible by rewinding V^* . Towards that goal, S employs a simple greedy strategy to incrementally generate and *fix* a partial view of V^* . Whenever V^* sends S a first stage message v_i , S checks if it had guessed correctly when committing to p_i . If so, S lengthens the partial view of V^* to include this correct guess. Otherwise, S rewinds V^* back to the previously generated partial view. This “incremental strategy” is somewhat reminiscent of [46], but since our protocol is public-coin, the actual analysis is quite different. Additionally,

⁷Here we use the following form of Chernoff bound. If $\{X_i\}$ are i.i.d. satisfying $\Pr[X_i = 0] = \Pr[X_i = 1] = 1/2$, then $\Pr[\sum_{i=1}^n X_i \geq n/2 + a] \leq e^{-2a^2/n}$

we take care to always simulate the stage two proof in a straight line fashion without rewinds, so that we may use a simple hybrid argument to show the zero-knowledge property.

We use superscripts to distinguish messages from different sessions. To prevent S from focusing too much on one particular session, we keep m counters, c^1, \dots, c^m , to record how much “work” has been done in each session. In general, S proceeds as follows to incrementally fix the view (originally the empty view is fixed). When asked to provide a prover message:

1. S commits to a fresh random bit for each stage one prover message.
2. For each stage two proof, S aborts if in this session, $p_i = v_i$ for less than $k^3 + k^2/2$ values of i . Otherwise, S uses this as a witness to generate the prover messages in the stage two proof.

When receiving a verifier message:

3. If S receives a message v_i^j (from session j) and $c^j < 2k^2$, it checks if the commitment to p_i^j is part of the fixed partial view. If yes, S simply continues, “giving up” on this guess. Otherwise, S checks if $p_i^j = v_i^j$. If yes, S extends the fixed partial view up to message v_i^j and increments c^j ; in this case we say v_i^j is *rigged*. If $p_i^j \neq v_i^j$, then S rewinds V^* to start a fresh continuation from the previously fixed partial view.
4. If S receives the second stage two verifier message from any session (e.g., the challenge message of the Blum Hamiltonicity protocol), it extends the fixed partial view up to the just received verifier message. As a consequence, all stage two proofs are simulated by S in a straight-line fashion without rewinds.

5. If S has performed $k - 1$ rewinds without rigging a message or encountering a stage two verifier message, and on the k^{th} try again receives $v_i^j \neq p_i^j$ where p_i^j is not fixed and $c^j < 2k^2$, S simply gives up and pretend to rig v_i^j anyway (albeit incorrectly). That is, S extends the fixed partial view up to message v_i^j and increments c^j .

The next two claims show that S is a k -bounded black-box zero-knowledge simulator when $k \in \omega(\log n)$.

Claim 39. *S runs in (strict) polynomial time.*

Proof. S performs at most $km(2k^2)$ rewinds, which is polynomial in n . □

Claim 40. *If $x \in L$ and $m \leq k$, $S^V(x, z)$ and $\text{View}_{V^*}^P(x, z)$ are computationally indistinguishable over n .*

Proof. We introduce a series of hybrids.

Hybrid 1. Our first hybrid S_1 is given witness w to the statement $x \in L$. S_1 proceeds identically as S until a stage two proof is reached. S_1 aborts if S aborts, but uses the witness w instead of the various p_i 's to complete the stage two proof. Even though S performs many rewinds, S never rewinds a partial stage two proof. Therefore, $S^V(x, z)$ and $S_1^V(x, z)$ are computationally indistinguishable because the stage two proof is witness indistinguishable.

Hybrid 2. Our second hybrid S_2 is identical to S_1 except that it samples two random bits for each stage one commitment p_i and q_i . S_2 commits to p_i , but checks v_i against q_i . Since S_1 gives polynomially many commitments and run

in polynomial time, and since each commitment is computationally hiding and independent from the rest of the execution of S_1 (stage two proofs are provided using w), $S_1^{V^*}(x, z)$ and $S_2^{V^*}(x, z)$ are computationally indistinguishable.

Hybrid 3. Our third hybrid S_3 is identical to S_2 except that S_3 *always* gives a stage two proof using witness w even if S_2 aborts. To see that $S_2^{V^*}(x, z)$ and $S_3^{V^*}(x, z)$ are computationally indistinguishable, it suffices to show that S_2 aborts with negligible probability.

Observe that whenever S extends the fixed partial view (either by rigging a commitment, or by encountering a verifier challenge in a stage two proof), at most one commitment from each session with less than $2k^2$ rigged messages is fixed as part of the simulator output. This is because before encountering a second commitment in any session, S would first try to rig the first commitment. For each session, S rigs at most $2k^2$ stage one commitments and encounter at most one stage two verifier challenge. Therefore, the number of commitments fixed per session without rigging is at most $(k - 1)(2k^2 + 1) = 2k^3 - (2k^2 - k + 1)$. In other words, every session will have at least $2k^2 - k + 1$ commitments rigged.

We now show that except with negligible probability, S_2 will have $k^3 + k^2/2$ correct guesses per session. Recall that the guesses of S_2 , q_i , are independent from V^* 's responses since these guesses play no part in the commitments sent to V^* . Therefore, except with probability $\text{poly}(n)2^{-k}$, every rigged commitment is a correct guess. Next, for the $2k^3 - (2k^2 - k + 1) \geq 2k^3 - 2k^2$ messages that are not rigged, we apply the Chernoff bound to see that except with probability $e^{-O(k)}$, we should have at least $(k^3 - k^2) - k^2/4 = k^3 - 5k^2/4$ correct guesses. Thus, except

with negligible probability⁸, we have a total of $(k^3 - 5k^2/4) + (2k^2 - k + 1) \geq k^3 + k^2/2$ correct guesses as desired.

Final step. S_3 is now identical to P (sends identically distributed messages) except that it may rewind V during the execution. But S_3 only rewinds if $q_i \neq v_i$, an event independent from the protocol execution. Therefore $S_3^{V^*}(x, z)$ is *identical* to $\text{View}_{V^*}^P(x, z)$. This concludes the proof. \square

4.6 A Non-Black-Box Public-Coin Parallel ZK Argument

4.6.1 The Protocol

Our non-black-box public-coin parallel zero-knowledge argument is similar to the non-black-box public-coin bounded-concurrent zero-knowledge argument of Barak [3]. Our argument PARALLELZKARG is described in Figure 4.3 which utilizes an additional relation \mathbf{R}_S defined in Figure 4.4. PARALLELZKARG is constant round, and can be based on collision resistant hash-functions. Intuitively, the zero-knowledge simulator will use a witness of the relation \mathbf{R}_S as a trapdoor.

In our construction, c denotes a vector, and c_{-i} denotes the same vector with the i^{th} element removed (i.e., c_{-1} is one shorter than c); $\{\mathcal{H}_n\}_n$ denotes a family of collision resistant hash-functions indexed by integer n ; $\{f_s\}_s$ denotes a family of pseudorandom functions indexed by $s \in \{0, 1\}^*$; and, $\text{Com}(x; r)$ denotes a statistically binding commitment of x using randomness r . We also make use of

⁸Recall again that 2^{-k} and $e^{-O(k)}$ are negligible in n since $k = \omega(\log n)$.

a witness-indistinguishable universal argument of knowledge, WI UARG [4], because the relation \mathbf{R}_S is quasi-polynomial time ($n^{\log n}$) instead of polynomial time.

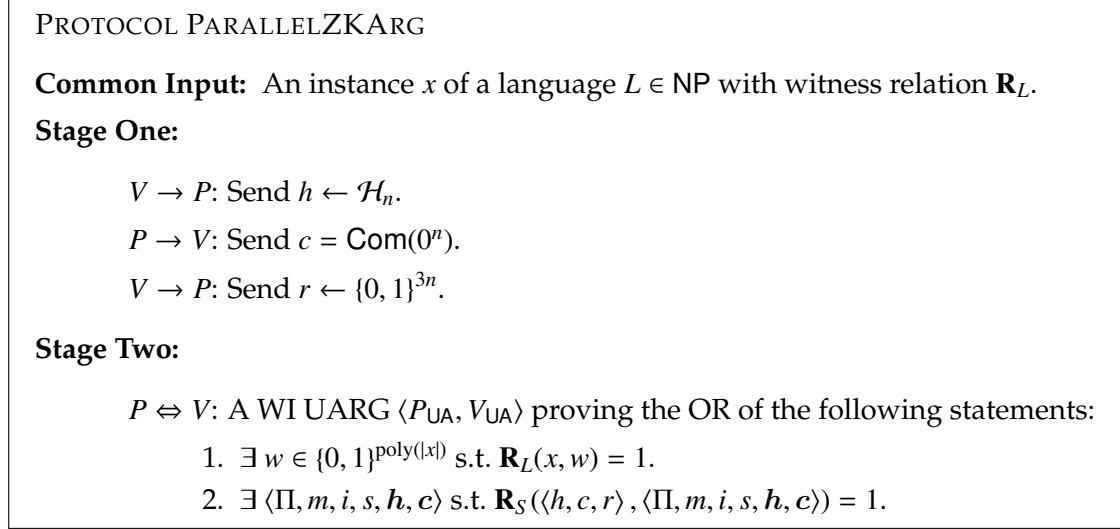


Figure 4.3: A public-coin non-black-box parallel zero-knowledge protocol.

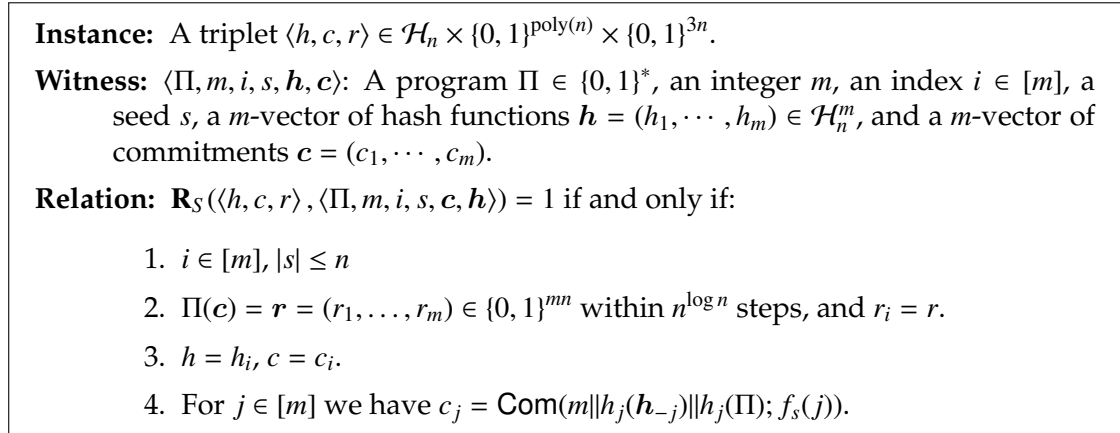


Figure 4.4: \mathbf{R}_S , an NP relation that extend Barak's construction [3] for parallel repetitions.

Simplifying Assumptions. We remark that the relation presented in Fig. 4.4 is slightly oversimplified and only works when $\{\mathcal{H}_n\}_n$ is collision resistant against

“slightly” super-polynomially sized circuits. To make it work assuming collision resistance against polynomially sized circuits, one should use a “good” error-correcting code ECC (i.e., with constant distance and with polynomial-time encoding and decoding), and replace the commitments $\text{Com}(m \| h_j(\mathbf{h}_{-j}) \| h_j(\Pi); f_s(j))$ with $\text{Com}(m \| h_j(\mathbf{h}_{-j}) \| h_j(\text{ECC}(\Pi)); f_s(j))$ [4]. We also assume that Com is a one-message commitment scheme. Such schemes can be constructed based on any one-to-one one-way function. At the cost of a small complication, the one-message scheme could have been replaced by the 2-message commitment scheme of [50], which can be based on any one-way function [35].

4.6.2 Completeness and Soundness

PARALLELZKARG is clearly complete; an honest prover can use the real witness to complete the Stage Two argument. The soundness of PARALLELZKARG follows from the same ideas as Barak’s non-black-box zero-knowledge protocol.

Lemma 41. *PARALLELZKARG has negligible soundness error against polynomial-time bounded provers.*

Proof. The main idea is that any deterministic program Π produces only one output for any given input. When the prover commits to c , it can no longer change the program Π , m or \mathbf{h} because Com is statistically binding and \mathcal{H}_n is collision resistant. Since we use a pseudo-random function to determine the randomness of the commitments, the value of $\Pi(c)$ would only depend on i and s . Since there are $m = \text{poly}(n)$ values of i and 2^n values of s , $\Pi(c)$ may take on at most $\text{poly}(n)2^n$ values after the commitment c is fixed. On the other hand, the verifier chooses $r \in \{0, 1\}^{3n}$ randomly after the prover fixes c . Therefore, the

probability that there exists some i and s so that the i^{th} component of $\Pi(c)$ is r is less than $\text{poly}(n)2^n/2^{3n} < 2^{-n}$ and is negligible.

We now prove the lemma formally. Suppose the contrary that some efficient cheating prover P^* breaks soundness of PARALLELZKARG with polynomial probability on an infinite sequence of inputs $\{x_n\}_n$, $x_n \in \{0, 1\}^n \setminus L$. Using P^* , we construct an adversary \mathcal{A} that acts either as a collision finder for \mathcal{H}_n , or a cheating sender for Com.

\mathcal{A} runs P^* internally. On input 1^n , \mathcal{A} starts by receiving a random $h \leftarrow \mathcal{H}_n$, and presents h to P^* as the first message of PARALLELZKARG; P^* responds by generating a commitment c . The following step is then repeated twice: \mathcal{A} sends a random challenge $r \in \{0, 1\}^{3n}$ to P^* , and uses the witness extractor of the Stage Two UARG on P^* to extract a (potentially quasi-polynomial-length) witness of the relation \mathbf{R}_S (we cannot extract a witness $w \in \mathbf{R}_L(x)$ since $x \notin L$). Because P^* breaks soundness with noticeable probability, \mathcal{A} succeeds in extracting two witnesses, $\langle \Pi, m, i, s, c, h \rangle$ and $\langle \Pi', m', i', s', c', h' \rangle$, to the statements $\langle h, c, r \rangle$ and $\langle h, c, r' \rangle$ (where r and r' are independent and uniform in $\{0, 1\}^{3n}$), also with noticeable probability. We split into two cases:

Case 1: $\Pi \neq \Pi'$ or $m \neq m'$ or $h_{-i} \neq h'_{-i'}$.

By the definition of \mathbf{R}_S , we have

$$\begin{aligned} c = c_i &= \text{Com}(m \| h(h_{-i}) \| h(\Pi); f_s(i)) \\ &= c'_{i'} = \text{Com}(m' \| h(h'_{-i'}) \| h(\Pi'); f_{s'}(i')) \end{aligned}$$

If $\Pi \neq \Pi'$, then either \mathcal{A} has found a collision to h (if $h(\Pi) = h(\Pi')$), or \mathcal{A} has broken the binding property of Com (by decommitting c to two different strings, $\cdot \| h(\Pi)$ and $\cdot \| h(\Pi')$). These conclusions hold in the same manner if

$$m \neq m' \text{ or } \mathbf{h}_{-i} \neq \mathbf{h}'_{-i'}.$$

Case 2: $\Pi = \Pi'$ and $m = m'$ and $\mathbf{h}_{-i} = \mathbf{h}'_{-i'}$.

In this case, given the first witness $\langle \Pi, m, i, s, \mathbf{c}, \mathbf{h} \rangle$, the value of $\Pi'(\mathbf{c}')_{i'}$ is fixed modulo the value of $i' \in [m]$ and $s' \in \{0, 1\}^n$; that is, $\Pi'(\mathbf{c}')_{i'}$ can take on at most $m \cdot 2^n$ values. However, $r' \in \{0, 1\}^{3n}$ is chosen independently from the first witness. Therefore $\Pi'(\mathbf{c}')_{i'} = r'$ (required for this case to occur) is possible with probability at most $m \cdot 2^n / 2^{3n} < 2^{-n}$.

Therefore we conclude that except with exponentially small probability (when Case 2 occurs), \mathcal{A} either finds a collision for h , or breaks the binding property of Com. This gives the desired contradiction. \square

4.6.3 Parallel ZK

Given a m -session parallel adversarial verifier V^* , we construct a (non-black-box) zero-knowledge simulator $S = S_{V^*}$ as follows. On input x and auxiliary input z , $S = S_{V^*}(x, z)$ first picks a random $s \in \{0, 1\}^n$ to fix the pseudorandom function f_s , and starts a straight-line simulation of V^* 's view. During the simulation, V^* starts by opening m sessions of PARALLELZKARG and sending hash functions h_i in session i . S is expected to respond in session i with some commitment c_i , after which V^* responds with a string r_i . The crux of the simulation is for S to commit to a program Π such that $\Pi(\mathbf{c}) = \mathbf{r}$ (including the case where some components of \mathbf{r} are aborts). But V^* is just such a program. Therefore, S sets Π to $V^*(x, z)$, and sets $c_i = \text{Com}(m \| h_i(\mathbf{h}_{-i}) \| h_i(\Pi); f_s(i))$. By construction, we now have for each session i , $\langle h_i, c_i, r_i \rangle, \langle \Pi, m, i, s, \mathbf{c}, \mathbf{h} \rangle \in \mathbf{R}_S$; this is because the messages $\mathbf{r} = (r_1, \dots, r_m)$ are indeed what V^* outputs given \mathbf{c} . These witnesses

allow S to complete the Stage Two arguments. After the simulation ends, S outputs the view of V^* during the simulation.

Clearly S runs in polynomial time. The following lemma establishes zero-knowledge.

Lemma 42. *The following ensembles are computationally indistinguishable over $n \in \mathbb{N}$:*

$$\{S_{V^*}(x, z)\}_{x \in L, z \in \{0,1\}^*} \approx \{\text{View}_{V^*}(P(x), V^*(x, z))\}_{x \in L, z \in \{0,1\}^*}$$

Proof. We use a simple hybrid argument.

Let S_1 be a simulator that is given a witness w for $x \in L$. S_1 proceeds as S but instead uses w to complete the Stage Two argument. By the witness-indistinguishable property of the Stage Two argument (and recall that witness-indistinguishability is preserved under parallel repetition [23]), the output of $S(x)$ and $S_1(x)$ are indistinguishable.

Let S_2 be the same simulator as S_1 except that S_2 uses fresh randomness for the commitments c_1, \dots, c_m . Since S_1 chooses s uniformly, f_s is a pseudorandom function, and both S_1 and S_2 are efficient, the output of $S_1(x)$ and $S_2(x)$ are indistinguishable. (Formally, this step can be further split into m hybrids; the i^{th} hybrid uses fresh randomness in the first i parallel sessions, and uses pseudorandomness in the other sessions.)

Let S_3 be the same simulator as S_2 except that S_3 commits to 0^n in the commitments c_1, \dots, c_m . By the computational-hiding property of Com , the output of $S_2(x)$ and $S_3(x)$ are indistinguishable. (Formally, this step can be further split into m hybrids; the i^{th} hybrid commits to 0^n in the first i parallel sessions, and commits following the strategy of S in the other sessions.)

But S_3 is identical to the honest prover, i.e., the output of $S_3(x)$ is identical to the view of V^* . Therefore, we have shown that the output of S is computationally indistinguishable from the view of V^* . \square

4.7 Application to Resetably-Sound Arguments

In this section we show how to achieve more general notions of resettable soundness that were not required for our main theorem. [26] implicitly shows that any constant-round public-coin argument is fixed-input resetably-sound if the verifier uses a pseudo-random function to generate its messages. [5, Proposition 3.5] extends the analysis to show that any constant-round public-coin argument of knowledge for $L \in \text{NP}$ is a (full-blown) resetably-sound argument of knowledge of L , again if the verifier uses a pseudo-random function to generate its messages. We give a pair of analogous theorems below, based on our techniques in Sect. 4.3.

Theorem 43. *Let $\Pi = (P, V)$ be a public-coin argument for an NP language L with negligible soundness error. Define $\tilde{\Pi}^m = \langle P^m, \tilde{V}^m \rangle$ to be m parallel repetitions of Π with the following modification: \tilde{V}^m will sample a pseudo-random function f at the beginning of the protocol, and construct each verifier message by applying f to the prover messages received so far. Then, whenever $m \geq k^2 \log^2 n$, $\tilde{\Pi}^m$ is a fixed-input resetably-sound argument.*

Theorem 44. *Let $\Pi = (P, V)$ be a public-coin argument of knowledge for an NP language L with negligible soundness error. Define $\tilde{\Pi}^m = \langle P^m, \tilde{V}^m \rangle$ similarly to Theorem 43. Then, whenever $m \geq k^2 \log^2 n$, $\tilde{\Pi}^m$ is a resetably-sound argument of knowledge.*

Note that in contrast with Sect. 4.3, we have replaced poly-wise independent hash-functions with pseudo-random functions. This is because a resettably-sound argument needs to guard against all polynomial-time resetting attacks, and so we cannot assume a universal bound on the running time of the attacks.

Proof sketch of Theorem 43. Suppose some polynomial time P_m^* breaks the fixed-input resettable-soundness property against \tilde{V}^m . Let \hat{V}^m be a hybrid verifier that is identical to \tilde{V}^m except that \hat{V}^m uses a truly random function F instead of a pseudo-random function f . Then, by the property of a pseudo-random function, P_m^* also breaks the fixed-input resettable-soundness property against \hat{V}^m . Now, the techniques of Sect. 4.3.3 shows how to construct a cheating P^* based on P_m^* that contradicts the soundness property of Π . This gives a contradiction. \square

Proof sketch of Theorem 44. We use the same techniques as [5]. Consider using the same proof sketch as Theorem 43. It is easy to extend the techniques of Sect. 4.3.3 to full-blown resettable attacks where P_m^* selects the input instances adaptively. The main subtlety, as pointed out by [5], is the hybrid argument involving the pseudo-random functions.

We need to show that if P_m^* breaks the resettable-soundness property against the pseudo-random \tilde{V}^m , then it should also break the resettable-soundness property against the truly random \hat{V}^m . The subtlety here is that a computationally-bounded distinguisher cannot determine whether P_m^* has completed a successful resetting attack or not, because it cannot determine whether the x 's chosen by P_m^* are in L or not. To overcome this obstacle, we require Π to be an argument of knowledge, i.e., there is a witness-extraction algorithm. We may then apply the

witness-extraction algorithm to P^* (constructed from P_m^*) to determine whether the input instance accepted by V are indeed in the language L or not. \square

BIBLIOGRAPHY

- [1] R. Axelrod. *The evolution of cooperation*. New York: Basic Books, 1984.
- [2] László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.
- [3] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS '01*, pages 106–115, 2001.
- [4] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Computational Complexity*, pages 162–171, 2002.
- [5] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resetably-sound zero-knowledge and its applications. In *FOCS'02*, pages 116–125, 2001.
- [6] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *STOC '02*, pages 484–493, 2002.
- [7] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS'06*, pages 345–354, 2006.
- [8] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, pages 390–420, 1992.
- [9] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *FOCS '97*, pages 374–383, 1997.
- [10] Manuel Blum. How to prove a theorem so no one else can claim it. *Proc. of the International Congress of Mathematicians*, pages 1444–1451, 1986.
- [11] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.
- [12] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO '01*, pages 19–40, 2001.

- [13] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC '00*, pages 235–244, 2000.
- [14] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. In *STOC '01*, pages 570–579, 2001.
- [15] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *J. Complex.*, 5(1):96–106, 1989.
- [16] Tzafrir Cohen, Joe Kilian, and Erez Petrank. Responsive round complexity and concurrent zero-knowledge. In *ASIACRYPT '01*, pages 422–441, 2001.
- [17] Ronald Cramer, Ivan Damgrd, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, pages 174–187, 1994.
- [18] Ivan Damgrd. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT '00*, pages 418–430, 2000.
- [19] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
- [20] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [21] Cynthia Dwork and Amit Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In *CRYPTO '98*, pages 177–190, 1998.
- [22] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *FOCS '90*, pages 308–317, 1990.
- [23] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90*, pages 416–426, 1990.
- [24] Oded Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [25] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *STOC '02*, pages 332–340, 2002.

- [26] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.
- [27] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.
- [28] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.
- [29] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7:1–32, 1994.
- [30] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [31] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [32] Vipul Goyal, Ryan Moriarty, Rafail Ostrovsky, and Amit Sahai. Concurrent statistical zero-knowledge arguments for NP from one way functions. In *ASIACRYPT '07*, pages 444–459, 2007.
- [33] Iftach Haitner, Minh-Huyen Nguyen, Shien Jin Ong, Omer Reingold, and Salil P. Vadhan. Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. *SIAM Journal of Computing*, 39(3):1153–1218, 2009.
- [34] Iftach Haitner, Alon Rosen, and Ronen Shaltiel. On the (im)possibility of Arthur-Merlin witness hiding protocols. In *TCC '09*, pages 220–237, 2009.
- [35] Johan Hstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.
- [36] Johan Hstad, Rafael Pass, Douglas Wikström, and Krzysztof Pietrzak. An efficient parallel repetition theorem. In *TCC '10*, pages 1–18, 2010.
- [37] Thomas Holenstein. Parallel repetition: simplifications and the no-signaling case. In *STOC '07*, pages 411–419, 2007.

- [38] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Chernoff-type direct product theorems. In *CRYPTO '07*, pages 500–516, 2007.
- [39] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the xor lemma. In *STOC '97*, pages 220–229, 1997.
- [40] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *STOC '05*, pages 644–653, 2005.
- [41] Jonathan Katz. Which languages have 4-round zero-knowledge proofs? In *Theory of Cryptography*, pages 73–88, 2008.
- [42] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithm rounds. In *STOC '01*, pages 560–569, 2001.
- [43] Joe Kilian, Erez Petrank, and Charles Rackoff. Lower bounds for zero knowledge on the internet. In *FOCS '98*, pages 484–492, 1998.
- [44] Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkatasubramanian. Concurrent non-malleable zero knowledge proofs. In *CRYPTO '10*, pages 429–446, 2010.
- [45] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkatasubramanian. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC '09*, pages 179–188, 2009.
- [46] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC '03*, pages 683–692, 2003.
- [47] Yehuda Lindell. Lower bounds for concurrent self composition. In *TCC '04*, pages 203–222, 2004.
- [48] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
- [49] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC '06*, pages 306–315, 2006.
- [50] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

- [51] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Precise concurrent zero knowledge. In *EUROCRYPT '08*, pages 397–414, 2008.
- [52] Rafael Pass. Parallel repetition of zero-knowledge proofs and the possibility of basing cryptography on NP-hardness. In *IEEE Conference on Computational Complexity*, pages 96–110, 2006.
- [53] Rafael Pass and Alon Rosen. New and improved constructions of nonmal-leable cryptographic protocols. *SIAM Journal on Computing*, 38(2):702–752, 2008.
- [54] Rafael Pass, Alon Rosen, and Wei-Lung Dustin Tseng. Public-coin parallel zero-knowledge for NP. In submission, 2011.
- [55] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent zero knowledge, revisited. In submission to *Journal of Cryptology*, 2011.
- [56] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Eye for an eye: Efficient concurrent zero-knowledge in the timing model. In *TCC '10*, pages 518–534, 2010.
- [57] Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikström. On the composition of public-coin zero-knowledge protocols. In *CRYPTO '09*, pages 160–176, 2009.
- [58] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. An efficient parallel repetition theorem for Arthur-Merlin games. In *STOC '07*, pages 420–429, 2007.
- [59] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In *TCC '08*, pages 553–570, 2008.
- [60] Giuseppe Persiano and Ivan Visconti. Single-prover concurrent zero knowledge in almost constant rounds. In *Automata, Languages and Programming*, pages 228–240, 2005.
- [61] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS '02*, pages 366–375, 2002.
- [62] Manoj Prabhakaran and Amit Sahai. Concurrent zero knowledge proofs

with logarithmic round complexity. Cryptology ePrint Archive, Report 2002/055, 2002. <http://eprint.iacr.org/2002/055>.

- [63] Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.
- [64] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 415–432, 1999.
- [65] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO '00*, pages 451–468, 2000.
- [66] Alon Rosen. *The round-complexity of black-box concurrent zero-knowledge*. PhD thesis, Weizmann Institute of Science, 2003.
- [67] Alon Rosen and abhi shelat. A rational defense against concurrent attacks. Manuscript, 2009.
- [68] Salil Vadhan. The complexity of zero knowledge. In *Foundations of Software Technology and Theoretical Computer Science '07*, pages 52–70, 2007.